Rochester Institute of Technology

Department of Networking, Security, and Systems Administration

Analysis of Aladdin Knowledge Systems' HASP HL and SafeNet's Sentinel Hardware Keys

Authors: Bradley Beam, Carmen De Vito, Andrew Gliddon, Duncan Grazier, Stephen Kent, Omar Khouri, Brian Luteran, Brian Meehan, John Migliaro, James Miller, Aaron Moulton, Edward Murrow, Seth Simons, John Tropiano, Ronald Valente

Editor: Matthew Burrough

Principal Investigator: William Stackpole

May 31, 2006

Disclaimer

Any access to or use of this Report is conditioned on the following:

1. The information in this Report is subject to change without notice.
2. The information in this Report is believed by its authors to be accurate and reliable, but is not guaranteed. All use of and reliance on this Report are at your sole risk. Neither this document's authors nor the Rochester Institute of Technology is liable or responsible for any damages, losses, or expenses arising from any error or omission in this Report.
3. **NO WARRANTIES, EXPRESS OR IMPLIED, ARE GIVEN BY THE AUTHORS OF THIS DOCUMENT OR THE ROCHESTER INSTITUTE OF TECHNOLOGY. ALL IMPLIED WARRANTIES, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT ARE DISCLAIMED AND EXCLUDED BY THE AUTHORS OF THIS DOCUMENT OR THE ROCHESTER INSTITUTE OF TECHNOLOGY. IN NO EVENT SHALL THE AUTHORS OF THIS DOCUMENT, OR THE ROCHESTER INSTITUTE OF TECHNOLOGY, BE LIABLE FOR ANY CONSEQUENTIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY LOSS OF PROFIT, REVENUE, DATA, COMPUTER PROGRAMS, OR OTHER ASSETS, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.**
4. This Report does not constitute an endorsement, recommendation or guarantee of any of the products (hardware or software) tested or the hardware and software used in testing the products. The testing does not guarantee that there are no errors or defects in the products, that the products will meet your expectations, requirements, needs, or specifications, or that they will operate without interruption.
5. This Report does not imply any endorsement, sponsorship, affiliation, or verification by or with any companies mentioned in this report.

All trademarks, service marks, and trade names used in this Report are the trademarks, service marks, and trade names of their respective owners, and no endorsement of, sponsorship of, affiliation with, or involvement in, any of the testing, this Report is implied, nor should it be inferred.

Table of Contents

Disclaimer **2**
Executive Summary **7**
Introduction **10**
Testing **11**
Section 1: Hardware Features **12**
Introduction 12
SafeNet Sentinel Hardware Key 12
Aladdin HASP HL 13
Common Features 14
Scoring 14
Section 2: Documentation **15**
Introduction 15
Features 15
Experiments to be performed 15
Interpretation of results 15
Experiment 16
Summary 17
Scoring 18
References 20
Sample Documentation Evaluation Form 21
Section 3: Protection Architecture **22**
Introduction 22
Features 22
Experiments to be performed 22
Interpretations of results 22
Experiment 1: Shell Execution Times 23
Experiment 2: Shell Comparisons 31
Experiment 3: Security Measures 32
Experiment 4: Encryption Levels 34
Experiment 5: Tiered Licensing 34
Experiment 6: On-Key Data Storage 35
Analysis 36
Summary 37
Scoring 37
References 38
Section 4: Remote Updates **39**
Introduction 39
Features 39
Experiments that were performed 39
Interpretation of Results 39
Methods of Update 40
Experiment 1: Transferring License Flat Files 41
Experiment 2: Complex License Flat File Transfer 42
Experiment 3: License Update to Several Clients 42

Analysis 43
Summary 44
Scoring 45
References 45
Section 5: Drivers and Libraries 46
Introduction 46
Experiment 1: Driver Availability 46
Experiment 2: Driver Installation (Windows XP) 50
Experiment 3: Microsoft Update 53
Experiment 4: Driver Certification 55
Experiment 5: Driver Redistribution 56
Scoring 57
Section 6: Developer & Customer Perspective 58
Introduction 58
Features 58
Relative Advantage for the Developer 59
Experiment 1.1: Comparison to Traditional Security Models 59
Complexity for the Developer 60
Experiment 2.1: Security by Shell Required Steps 60
Experiment 2.2: Security by API Required Steps 62
Experiment 2.3: Multiple Application Security 63
Compatibility for Developers 64
Experiment 3.1: Driver Installation/Removal 64
Trialability 64
Experiment 4.1: SDK Procurement 64
Experiment 4.2: Support Options 65
Analysis 66
Observability for the Developer 69
Experiment 5.1: Replacement Tokens 69
Relative Advantage for the End User 70
Experiment 1.1: Comparison to Traditional Security Models 70
Compatibility for the End User 71
Experiment 2.1: Runtime Limitations 71
Complexity for the End User 75
Experiment 3.1: User Interaction 75
Trialability for the End User 75
Experiment 4.1: Trial Scope of Features 75
Scoring 87
Section 7: Application Programming Interface 89
Introduction 89
Features 89
Experiments to be performed 89
Interpretation of results 90
Experiment 1: Language Support 90
Experiment 2: API Toolbox Ease-Of-Use 92
Experiment 3: Method Consistency between Languages 93

Summary 96
Scoring 96
References 97
Section 8: Vendor Tools 98
Introduction 98
Features 98
Experiments to be performed 98
Experiment 1: User Interface 99
Experiment 2: Toolbox Functionality 106
Experiment 3: Code Generation Facilities 108
Experiment 4: Sample Code Quality 110
Analysis 112
Scoring 112
References 112
Section 9: Security 113
Introduction 113
Features 113
Experiments performed 114
Experiment 1: Evaluation of Wrapper Protection 114
Part A) Comparison of wrapper strength - Size 114
Part B) Comparison of wrapper strength – Susceptibility to Attack 116
Experiment 2: Program – USB key interaction 117
Experiment 3: Replay Attacks 123
Scoring 123
References 123
Section 10: Licensing Models 125
Introduction 125
Features 125
Experiments to be performed 125
Interpretation of results 125
Experiment 1: Number of Keys that can be Programmed 126
Experiment 2: Number of Applications per User Key 126
Experiment 3: Is There a Way to Create a Custom License? 126
Experiment 4: License Activation/Deactivation in field 127
Experiment 5: API and Licensing Model Correlation 127
Analysis 128
Summary 128
Scoring 128
References 128
Section 11: Protection Features 129
Introduction 129
Features 129
Experiments performed 129
Interpretation of results 129
Experiment 1: 32-bit Portable Applications 129
Experiment 2: Windows DLLs 131

Experiment 3: .NET Testing 132
Experiment 4: Removing Key during Application Execution 133
Summary 134
Scoring 136
References 136
Conclusion 137
Appendix A – Terms 139

Executive Summary

At the request of the SafeNet Corporation, researchers at the Rochester Institute of Technology compared SafeNet's Sentinel hardware key to the Aladdin HASP HL. The goal of this research was to compare the features of each product and evaluate their advantages and disadvantages. While the research request was made by SafeNet, the researchers remained neutral in their examination.

The features of each product suite were analyzed so that a common model of comparison could be deduced. Researchers were assigned to the following aspects of the two products:

- Documentation
- Protection Architecture
- Remote Updates
- Drivers and Libraries
- Developer & Customer Perspective
- Application Programming Interface
- Vendor Tools
- Security
- Licensing Models

The documentation provided by both companies proved to be reasonably organized and contained much of the relevant information needed by users and developers alike. However the Aladdin HASP HL documentation received a higher grade due to its overall better quality of writing.

SafeNet's Sentinel Hardware Key fared better in the Protection Architecture analysis. Researchers in this category found the shell/wrapper code to be more intertwined into the protected application, and therefore less likely to be thwarted. It was also noted that the Aladdin HASP HL relies more heavily on its API library.

The team researching Remote Updates found that both vendors took different approaches to providing additional licensing once a product has been distributed. Aladdin's HASP HL scored just marginally higher with its easy-to-use wizard which customizes the remote update tool. The researchers commented on the lack of delegation to third party distributors for both vendors.

The drivers and libraries included with the sample toolkits from both vendors allowed full functionality of the hardware tokens. However, Aladdin's HASP HL earned a higher score in this category for providing a wider range of supported platforms, a more full-featured driver installer, Windows HQL certification, and distribution via Microsoft Update.

The team which focused on Customer & Developer Perspective applied Everett Roger's principles of Diffusion of Innovations to the products in the study. Both vendors'

hardware tokens provided a distinct relative advantage over traditional models of software protection. The shell/wrapper tools were not overly complex to implement, though the SafeNet Sentinel could be applied in nearly half the steps of its competitor. The opposite was true for the implementation of Sentinel API toolkit, however. Both products were found to be quite compatible with the host platforms used, however support for Windows Terminal services was problematic for both vendors. Researchers were able to learn that replacement keys are readily available and can be deployed effectively with the tools provided in the sample kits, though pricing details were limited. Support, however, was the one factor that earned the Aladdin HASP HL a higher score in this category than the SafeNet Sentinel. The website assistance from SafeNet, as well as telephone support, was inadequate or incorrect when queried.

Researchers implementing the Application Programming Interface (API) found a well supported platform of base languages, with additional support for relatively more obscure development environments in the Aladdin camp. The analysts would have liked to see more support given to the API toolkits, in the same manner as the wizards provide assistance with shell/wrapper technology. The SafeNet product earned higher marks for its consistency of method calls between languages, providing a seamless transition for developers using hardware token technology in multiple languages.

The Vendor Tools team found a very evenly matched comparison between the two products. The experience of the toolkits gave well formed code samples and an easy-to-implement shell/wrapper. The team awarded an edge to the SafeNet Sentinel for its simplified approach to wrapper implementation.

The Security review examined the way that the hardware tokens communicated with the protected applications. The complex algorithms incorporated by both products ruled out the use of any brute force tactics. They also were able to combat replay attacks. The SafeNet Sentinel Hardware key used multiple encryption methods and obfuscated its wrapper code, thus earning a higher score.

The products from both vendors earned an equal score from the Licensing team. Their research showed that both products allowed coarse licensing control with a shell or wrapper, and finer control with specific API methods. Additionally, the keys Sentinel S and the HASP HL Max both allow developers to license multiple applications on a single key, up to 112.

In terms of Protection Features, our researchers found that, while both products were able to successfully create secure shells around Microsoft Windows 32-bit portable applications, the SafeNet Sentinel Hardware Key toolkit failed to wrap some DLL files. The DLL files became corrupted and caused the application to fail, however the appropriate security messages detecting the key still displayed. The researchers also noted that the Aladdin HASP HL toolkit was unable protect Microsoft .NET 2.0 executables and appropriate error messages were displayed which indicated that the platform was unsupported.

The authors of this document advise the readers to carefully review each section and evaluate the merits the two hardware tokens according to their individual needs.

Introduction

The purpose of this report is to compare Aladdin's HASP HL and SafeNet's Sentinel USB keys. These keys aim to eliminate software piracy by requiring a user to insert a physical device into the computer before a given application can start. In order to compare these products thoroughly, a number of tests were designed and run against keys from each manufacturer. Testing also reached beyond the physical devices – calls were placed to each company's support line to test their service; documentation was reviewed for accuracy and readability, and software that accompanies the devices was examined.

Both products performed well and each had their advantages and disadvantages. Since a feature that is crucial for one application may be superfluous for another, it would be advisable to look at findings throughout the report and decide which features are most important for one's target application.

While the research team feels these tests were fair, accurate, and balanced, no test is perfect. As such, readers are encouraged to review these findings and conduct their own testing as they see fit. Different businesses have different requirements for products. The tests examined only some of these issues and were limited to those the researchers thought most important.

Testing

In order to compare the Aladdin HASP HL to the SafeNet Sentinel, a series of eleven testing areas have been established. These areas are:

- API
- Developer/Customer Perspective
- Documentation
- Drivers and Libraries
- Hardware Features
- Licensing Models
- Protection Architecture
- Protection Features
- Remote Update
- Security
- Vendor Tools

A team of researchers has analyzed each testing area. Their findings are published in the following sections.

Section 1: Hardware Features

Introduction

This section addresses the hardware from each vendor and how it interfaces with a computer. Features such as bus speed, memory capacity, and the architecture of the keys will be examined. For each product, the user keys as well as the master key will be tested. This will include reviews of features not mentioned within the documentation for each product. These features will be explored through various experiments with diagnostic software and hardware. During the course of testing, the keys will not be disassembled or otherwise modified on a physical level.

One of the main features advertised by Aladdin and SafeNet is the ability to detect listening software. However, the encrypted traffic between the computer and the keys was recordable. Traffic from the SafeNet Sentinel's user, developer, and distributor keys, as well as Aladdin's developer and user keys, was captured without difficulty. This is the first step needed to defeat the key's encryption algorithm and possibly compromise the protection the key provides.

SafeNet Sentinel Hardware Key

(http://www.ebizlatam.com/imagenes/noticias/equipos/sentinel/ikey-with-lid.jpg)

- Size: 2 2/16"x 9/16" x 5/16" (all measurements taken at the widest points)
- Key ring loop
- LED indicator to indicate successful connection to the computer

The Sentinel Hardware Key offers several features over the Aladdin, one feature being what SafeNet calls a V-Clock. This V-Clock is a way for the key to keep track of time-based licenses, meaning a license based on a certain amount of time: a day, week, month, or other specified length of time. The V-Clock is defined as a continuous snapshot of the clock and date as the User Key is plugged in. When the key is plugged back into the computer, the time is compared to the V-Clock. If enough time has elapsed, the license expires and the program cannot be run. Otherwise, the V-Clock keeps track of the time left on the license. SafeNet states that if there is an extreme deviation of the date and time, say if one were to roll the clock on the computer back, the V-Clock can sense this and terminate the license. Sentinel is offering a new option that replaces the V-Clock,

and gives the key its own clock, dubbed the Real Time Clock (RTC), which keeps track of the time on a license.

Traffic traveling between the hardware key and the programs that are protected by the Sentinel's Shell should be another path of attack. Sentinel states the traffic is encrypted, meaning the contents are changed in such a way that the sender and receiver are the only parties able to read the information. This encryption, Advanced Encryption Standard (AES), uses a 128-bit key, but as an option, you can utilize another layer of encryption, Elliptic Curve Cryptosystem (ECC), with a 163-bit key.

Aladdin HASP HL

(http://www.aladdin.de/images/i_bilder_grafiken/hasp_hl_net_rot_w300.jpg)

- Size: HASP HL Max Size 39 x 53 x 17 mm
- Key Ring Loop

The Aladdin HASP HL comes in several models – Basic, Pro, Max, Time, and Net. All models except the Basic offer unique ID numbers and have internal memory. Max, Time, and Net models offer 4KB of memory and are designed to protect up to 112 applications. The Pro model has 112 Bytes of memory and protects up to 16 applications. The Basic model has no internal memory or unique id number and can only support one application. The version tested in this paper is the HASP HL Max, which comes with 4KB of memory and has support up to 112 licenses at one time. The estimated duration of memory retention is ten years and the memory cell can be rewritten at least 1,000,000 times.

The HASP HL Time keys come with an onboard clock called a Real Time Clock. This clock allows a key to track the time continually. It has a 4-year battery life, although it can be extended up to 10 years by connecting it to a functioning computer. If the battery dies, the HASP HL Time can function as a HASP HL Max and does not support time-based licensing, but then uses activation-based licensing instead.

The HASP HL Max uses a 128-bit AES symmetric encryption engine. The symmetric encryption engine allows the HASP HL Max to protect the data being transmitted and decrypt data it received.

Common Features

Both hardware-based tokens state they can detect a third party listening on the bus where the USB resides. However, the reviewers were able to view the actual traffic between both tokens during the various stages the key is accessed. The traffic between the computer and the Distributor Key for SafeNet and the Master Key for Aladdin was also viewable. Although traffic is encrypted using 128-bit AES, it is still the first step attackers must make when looking for weaknesses.

Another feature that both HASP HL and SafeNet Sentinel offer is access to a real-time clock on the keys. The SafeNet Sentinel offers the V-Clock, which keeps track of time-based licenses. The V-Clock allows the user to reliably and securely offer time-based license models such as trial, demo, or subscription. HASP HL Time contains an internal real-time clock, indicating the exact time (hours, minutes, and seconds) and date (day, month, and year). Specifically designed to enable software renting or leasing, the HASP HL Time lets software distributors charge their clients periodically for software use and maintenance.

Scoring

Hardware Features	**Aladdin** HASP HL	**SafeNet** Sentinel Hardware Key
Overall	**100**	**100**

The features discussed in this section are examined throughout this document; therefore, since all features were found and validated, each received a perfect score.

Section 2: Documentation

Introduction

This section addresses the documentation provided with both the SafeNet Sentinel Hardware Key and the Aladdin HASP HL. Each chapter of the documentation was reviewed by a team of researchers and was graded based on the following sections: manuscript organization, section titles, depth of topic covered, quality of the writing, relevance of information being provided, and the clarity of the information being provided.

Features

Both SafeNet and Aladdin offer lengthy documentation – each document is over 200 pages long. SafeNet includes two appendices before their index. The first appendix to the SafeNet documentation is a glossary of terms. The second appendix is a list of hardware specifications of the Sentinel S standard key. Aladdin includes three appendices before its index. The first appendix is troubleshooting; the second is HASP demo passwords, while the third is a list of HASP keys and hardware specifications for each key type. Aladdin also includes a HASP glossary before its index.

Experiments to be performed

The team of reviewers graded the two documents on a one-hundred-point scale for six areas, the first being manuscript organization. Here, the team looked to see if the information was presented in a logical sense. The second area reviewed was the section titles. In this section, the team looked at the titles of each section within the chapter to determine if the title accurately represented the contents of the section. The next area examined was the depth of the topic that was being covered in the chapter. The reviewers were interested in whether or not there was a sufficient amount of information within the chapter, and that the chapter was able to answer any questions that may have arisen while reading the chapter. The fourth area that the reviewers evaluated was the quality of the writing in terms of grammar and spelling. The fifth area that was examined by the reviewers was the relevance of information being provided; essentially, did the chapter maintain its focus? The final area covered by the review team was to determine whether the information in the chapter was presented in a clear manner and if it was easy to understand. To maintain fairness while reviewing the documents, one reviewer read the SafeNet Sentinel Hardware Key documentation first, followed by the Aladdin HASP HL documentation. The other reviewer read the Aladdin HASP HL documentation first, followed by the SafeNet Sentinel Hardware Key documentation. This helped to eliminate first-read bias.

Interpretation of results

In order to collect data on the quality of the documentation, the review team printed out the Sentinel Developer's Guide version 1.0 and the HASP Programmer's Guide version

12. Then the review team determined what scoring rubric would be used. Finally, the review team determined who would complete specific tasks, and then agreed not to discuss his findings until he had finished his review in order to avoid biasing other's results.

The review team analyzed the documentation of several categories. Each category was initially assigned a total score of 100 out of 100 points. Five points were then deducted for each spelling mistake, grammatical error, or other failure noted by the reviewer. Scores were then calculated for each category in each chapter. Finally, total average scores were calculated for each document.

Experiment

Manuscript Organization

SafeNet: SafeNet's documentation was laid out logically, beginning with a definition of piracy, followed by an overview of its key, its features, and how it could be implemented. The document then became more in depth about API functionality, remote update capabilities, and other implementation-specific issues.

Aladdin: Aladdin's documentation was laid out logically, beginning with what HASP can do and why companies should implement this level of protection. The document then discussed HASP features at length, including API functionality, remote update capabilities, and the differences in implementing their various key types.

Section Titles

SafeNet: After reading each section, each reviewer felt that the title adequately represented the topic material covered in that section.

Aladdin: After reading each section, each reviewer felt that the title adequately represented the topic material covered in that section.

Depth of Topic Covered

SafeNet: After reading the entire document, points were deducted for the omission of a troubleshooting appendix and for several terms that were introduced without being explained. Specifically, "Personal Folder" (page 176) and "LGX file type" (page 22) were not explained. In addition, the terms "toolkit" (page 10) and "shell" (page 6) are used without any introduction or discussion. The use of a Frequently Asked Questions section at the end of each chapter was a bonus. Additionally, SafeNet's documentation broke down required steps, recommended steps, and optional steps very well.

Aladdin: After reading the entire document, points were deducted for not indicating how much hard disk space was needed on page 52. In addition, points were deducted as both reviewers felt that some topics were over-covered. In several places, entire paragraphs were repeated with only small changes. Rather than lengthening the document

significantly, Aladdin might consider revising its process descriptions to discuss various configurations more succinctly.

Quality of Writing

SafeNet: The reviewers found multiple spelling and grammatical errors (see table one for a complete list of errors).

Aladdin: The reviewers found no spelling mistakes, and found only a few instances where sentence structure was grammatically questionable.

Relevance of Information Provided

SafeNet: Neither reviewer felt that any information provided was irrelevant to the topic being discussed.

Aladdin: Neither reviewer felt that any information provided was irrelevant to the topic being discussed.

Clarity of Information Provided

SafeNet: Screenshots and other images provided in the documentation were not clear when viewed as PDFs or in print. Additionally, some grammatical errors were so disruptive that certain paragraphs became unclear.

Aladdin: Screenshots and other images provided in the documentation were not clear when viewed as PDFs or in print. Additionally, some grammatical errors were so disruptive that certain paragraphs became unclear.

Summary

Based on the evaluation performed above, the reviewers felt that the Aladdin documentation was more complete, readable, and overall, more useful. This may be due to the higher revision number of Aladdin's document. SafeNet's spelling and grammatical errors greatly detracted from its readability. Additionally, the reviewers felt that the Aladdin documentation was more professionally written than SafeNet's. This opinion was influenced heavily by word choice in Aladdin's documentation. Both companies would have significantly increased scores if grammatical errors were removed, better screen shots were used, and overall clarity was increased.

Scoring

Documentation	**Aladdin** HASP HL	**SafeNet** Sentinel Hardware Key
Manuscript Organization	100	100
Section Titles	100	100
Depth of Topic Covered	95	90
Quality of Writing	95	68
Relevance of Information Provided	100	100
Clarity of Information Provided	85	90
Overall	**95.83**	**91.33**

Table 2.1: Errors found in SafeNet Documentation

Error	*Location*
"At the most basic level, your application is binded"	page 4 the first sentence under "Sentinel Keys Protect Against Piracy"
"It is a is a challenge-response"	Page 7 "Cutting-Edge Anti-Piracy Technology" - 3rd bullet
"The feature template group approach of toolkit".. (the Toolkit?)	Page 10 - "Smart & Flexible (One-Time) Implementation" -1st bullet
"Allow license sharing for..." The	page 10 under "Licensing Options for Increased Market Penetration" 6th bullet(last one)
"A web browser based tool using which the..."	Page 15 – Summary of Sentinel Key SDK
"that can used by"	Page 15 – Summary of Sentinel Key SDK
"using which you can"	Page 16 – Sentinel Keys Toolkit Section
Where do I find it	Page 26 First line see hard limit
"can stand-lone"	Page 32 – FAQ Question 1
"Refer to next section page 39..."	Page 36 Shell Feature Section
"License Designer screen of toolkit"	Page 37 First Note
"...template can has multiple"	Page 37 After first note
"Toolkit"	Page 37 Second note
"...Shell is capable of detecting the debuggers like"	Page 40 Anti-debugging Measures
"Click the b tab in the Add Shell Feature dialog box"	Page 62 – Providing Security Settings
"Only developer can generate"	Page 99 - 2nd Note
"...what all files"	Page 110 – FAQs Question 4
"Creating New Group"	Page 131
"Loading Group"	Page 132
"Duplicating Group"	Page 133
"Removing Group"	Page 133
"Viewing Group Layout"	Page 134
"Modifying Default Feature Instance"	Page 135

Error	*Location*
“Creating New Feature Instance”	Page 135
“Locking Group”	Page 137
“Unlocking Group”	Page 138
“It can happen so when the license template was updated”	Page 142
“Using which, they can program...”	Page 147
“...unintended recipients/process”	pg 173 – Encryption
“multiple of minutes”	pg 174 – Expiration Time
“license designer screen at the time of while adding/editing...”	pg 175 – Feature Instance

References

SafeNet Inc. SafeNet Sentinel Developer's Guide. SafeNet Inc. Baltimore. 2005.
Aladdin Inc. HASP Programmers Guide v1.3 Aladdin Knowledge Systems Ltd. 1985.

Sample Documentation Evaluation Form

Company/Chapter _____
Reviewer's Name _____

Criteria: one per chapter

Content

Manuscript Organization ____/100
Does the information presented make sense in the order it is presented?
C**omments (why you graded the way you did):**

Title accurately reflects content ____/100
If you had to determine the contents of this by looking at the title only, could you? If you opened to the table of contents, would you know this chapter had the information you wanted?
Comments (why you graded the way you did):

Topic area covered in sufficient depth ____/100
Do you have questions that were raised in the chapter, but never answered?
Comments (why you graded the way you did):

Quality of Writing (spelling, grammar) ____/100
Comments (why you graded the way you did):

Relevant information ____/100
Is the information provided needed? Are you being told how to use the product, or what the old version used to do?
Comments (why you graded the way you did):

Clarity ____/100
Was the information presented in a clear and helpful manner? Were you able to read the chapter and understand all of what you read?
Comments (why you graded the way you did):

Section 3: Protection Architecture

Introduction

In this section of the report, the protection architecture of both Aladdin's HASP HL key system and the SafeNet Sentinel key are compared. Many of the comparisons are based on the researchers' interpretation of the documentation. This is due to the inability to directly experiment and compare some features. The majority of this section is based on knowledge acquired from the documentation of the HASP HL and SafeNet Sentinel keys, as well as that from a previously published document comparing two of the older versions of these products.

Features

The current feature set reviewed in this section includes elements from other sections of this document including Licensing, Protection Features, Security, API, and Vendor Tools. Research has been performed on the following topics:

- Effect of Encryption and Shell Levels on Program Performance
- Methods of Encrypting and Protecting USB Traffic between Host and Key
- Method of Data Storage on the Hardware Keys
- Encryption Levels – Differences, Implementations
- Implementation of Shell Levels
- Licensing System

Experiments to be performed

This section is comprised of qualitative research into the different methods used by the hardware keys to protect software. The key areas that were examined are the same as those features stated above in the Features section.

One of the quantitative experiments performed by the researchers included a time trial to observe the effects of different levels of protection against the performance and size of a targeted application. In order to simplify the process, a simple "Hello World" control program was used in this experiment. Please note that the term shell or shelling is in relation to both the shell system used in the Sentinel keys as well as the Envelope system used in the HASP keys.

Interpretations of results

Data collection was performed by the researchers reading both the official documentations for the dongles and third party references to the systems. In addition, data was collected on the Time Trials section via quantitative experimentation. The scoring system for each experiment was based on a 100-point scale. The mean of the scores in this section was calculated.

Experiment 1: Shell Execution Times

Description

This experiment was designed to discover if there was a relationship between the time that it takes a program to execute and the shell level used to protect it. In addition, this experiment was designed to develop a better understanding of the shell's effect on the size of the executable after being protected.

Experimental Design

In order to accurately experiment and measure latency time between the time the program is executed and when it is actually active, a combination of scripts and a simple "hello world" program was used. The experiment consists of a script that attempts to run the shelled program and record the time elapsed between start and finish. The script stores this data in a simple tab delimited text file to simplify the analysis process. In order to accurately experiment and compare the two products, a shell/envelope scale was created to simplify the process:

Level	Sentinel Level	HASP Level
None	0	0
1	1	10
2	2	20
3	3	30
4	4	40
5	5	50

Each shell was created using factory defaults, except for shell level parameters. In order to retrieve accurate data, each shell level was examined 10 consecutive times and the results were averaged. These were then defined with the following naming convention: HASP: test_HASP#.exe (note that the encryption levels are directory names), and Sentinel: test_SAFE#.exe

In order to correctly experiment and compare the two products, a sterile and consistent system environment has been chosen for each experiment in the section. The environment consists of a Windows XP system with SP1 installed.

Data

HASP

	ENC1	ENC2	ENC3	ENC4	ENC5
None	0.028125	0.028125	0.029688	0.029688	0.029688
Level 10	0.740625	0.793750	0.782813	0.729688	0.734375
Level 20	0.809375	0.887500	0.898438	0.889063	0.885938
Level 30	0.831250	0.893750	0.887500	0.881250	0.876563
Level 40	0.737500	1.121875	1.120313	1.092188	1.112500
Level 50	1.089063	2.281250	2.273438	2.256250	2.268750

Figure 1.1 – HASP Time Trial Data (Note: ENC stands for Encryption level)

Sentinel

	Average
None	0.031250
Level 1	3.528125
Level 2	4.517188
Level 3	5.787500
Level 4	7.151563
Level 5	9.370313

Figure 1.2 – Sentinel Time Trial Data

```
Dim st
Dim en
Dim final1
Dim final2
Dim final3
Dim final4
Dim final5
Dim final6
Dim final7
Dim final8
Dim final9
Dim final10
Dim WSHShell
Set WSHShell = WScript.CreateObject("WScript.Shell")

        st = Timer
        WSHShell.Run WScript.Arguments.Item(0), 2, true
        en = Timer
        final1 = en - st

        st = Timer
        WSHShell.Run WScript.Arguments.Item(0), 2, true
        en = Timer
        final2 = en - st
```

```
	st = Timer
	WSHShell.Run WScript.Arguments.Item(0), 2, true
	en = Timer
	final3 = en - st

	st = Timer
	WSHShell.Run WScript.Arguments.Item(0), 2, true
	en = Timer
	final4 = en - st

	st = Timer
	WSHShell.Run WScript.Arguments.Item(0), 2, true
	en = Timer
	final5 = en - st

	st = Timer
	WSHShell.Run WScript.Arguments.Item(0), 2, true
	en = Timer
	final6 = en - st

	st = Timer
	WSHShell.Run WScript.Arguments.Item(0), 2, true
	en = Timer
	final7 = en - st

	st = Timer
	WSHShell.Run WScript.Arguments.Item(0), 2, true
	en = Timer
	final8 = en - st

	st = Timer
	WSHShell.Run WScript.Arguments.Item(0), 2, true
	en = Timer
	final9 = en - st

	st = Timer
	WSHShell.Run WScript.Arguments.Item(0), 2, true
	en = Timer
	final10 = en - st

WScript.Echo final1 & vbTab & final2 & vbTab & final3 & vbTab & final4 & vbTab &
final5 & vbTab & final6 & vbTab & final7 & vbTab & final8 & vbTab & final9 & vbTab
& final10
```

Figure 1.3 – Time trial script (VBScript)

```
@echo off
cscript //NOLOGO testCycle.vbs test_ORIGINAL.exe

cscript //NOLOGO testCycle.vbs test_SAFE1.exe
cscript //NOLOGO testCycle.vbs test_SAFE2.exe
cscript //NOLOGO testCycle.vbs test_SAFE3.exe
cscript //NOLOGO testCycle.vbs test_SAFE4.exe
cscript //NOLOGO testCycle.vbs test_SAFE5.exe
```

Figure 1.4 – Sentinel-specific trial script (batch)

```
@echo off
cscript //NOLOGO testCycle.vbs test_ORIGINAL.exe

cscript //NOLOGO testCycle.vbs test_HASP1.exe
cscript //NOLOGO testCycle.vbs test_HASP2.exe
cscript //NOLOGO testCycle.vbs test_HASP3.exe
cscript //NOLOGO testCycle.vbs test_HASP4.exe
cscript //NOLOGO testCycle.vbs test_HASP5.exe
```

Figure 1.5 – HASP-specific trial script (batch)

Results

Based on the data gathered from the experiment, a number of conclusions can be drawn. The most obvious of these is that the shell level has a direct impact on both the size of the file as well as program execution latency. It seems that as the level of shelling increases, the time it takes to execute the shell also increases. This shows that additional security precautions are added into the shelled programs, as both the SafeNet and Aladdin keys stated in their documentation. It appears that the changes between the shell levels are linear in the Sentinel system while, in the HASP system, the change has a nonlinear progression.

It appears that the change in the size of the shelled program is greater for the Sentinel keys compared to the HASP key. This could be caused by a difference in shell design in either system or one shell may use additional security precautions.

The devices were scored as follows:

	HASP	Sentinel
File Sizes (Overall)	9/13 (69.23%)	1/13 (7.69%)
<= 0.5 MB	0	0
<= 1.0 MB	0	0
<= 1.5 MB	0	0
<= 2.0 MB	0	0
<= 2.5 MB	1	0
<= 3.0 MB	1	0
<= 3.5 MB	1	0
<= 4.0 MB	1	0
<= 4.5 MB	1	0
<= 5.0 MB	1	0
<= 5.5 MB	1	0
<= 6.0 MB	1	0
<= 6.5 MB	1	1
Latency (Overall)	5/5 (100.00%)	1/5 (20.00%)
<= 2 sec	1	0
<= 4 sec	1	0
<= 6 sec	1	0
<= 8 sec	1	0
<= 10 sec	1	1
Total	84.62 %	27.69%

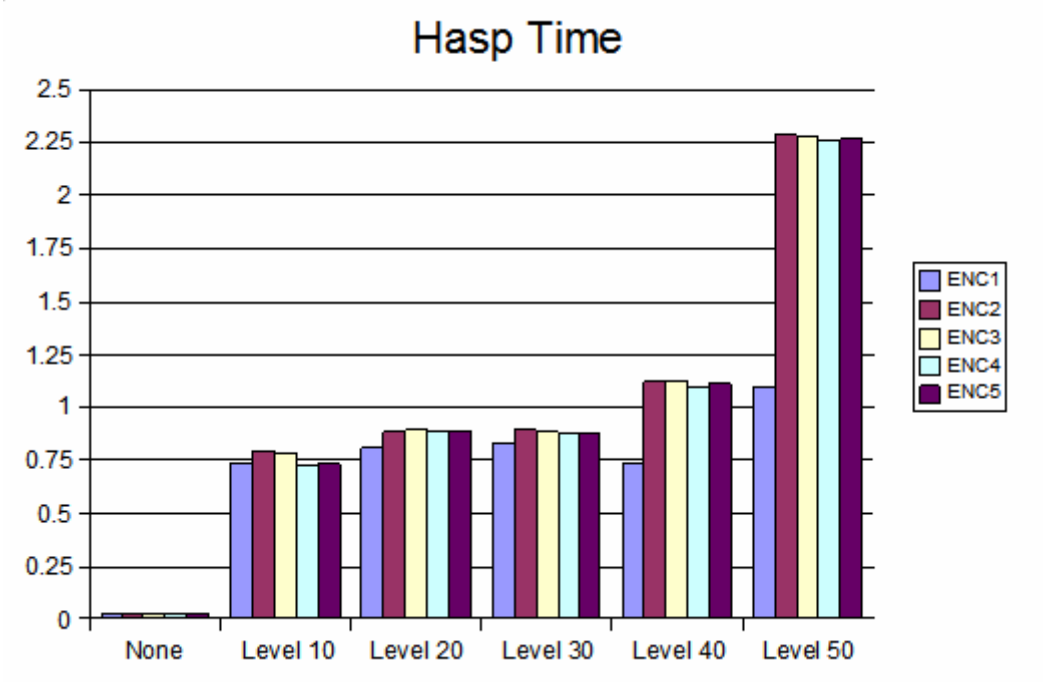


Figure 1.6 – Graph of HASP time trial (in seconds) (Lower is better)

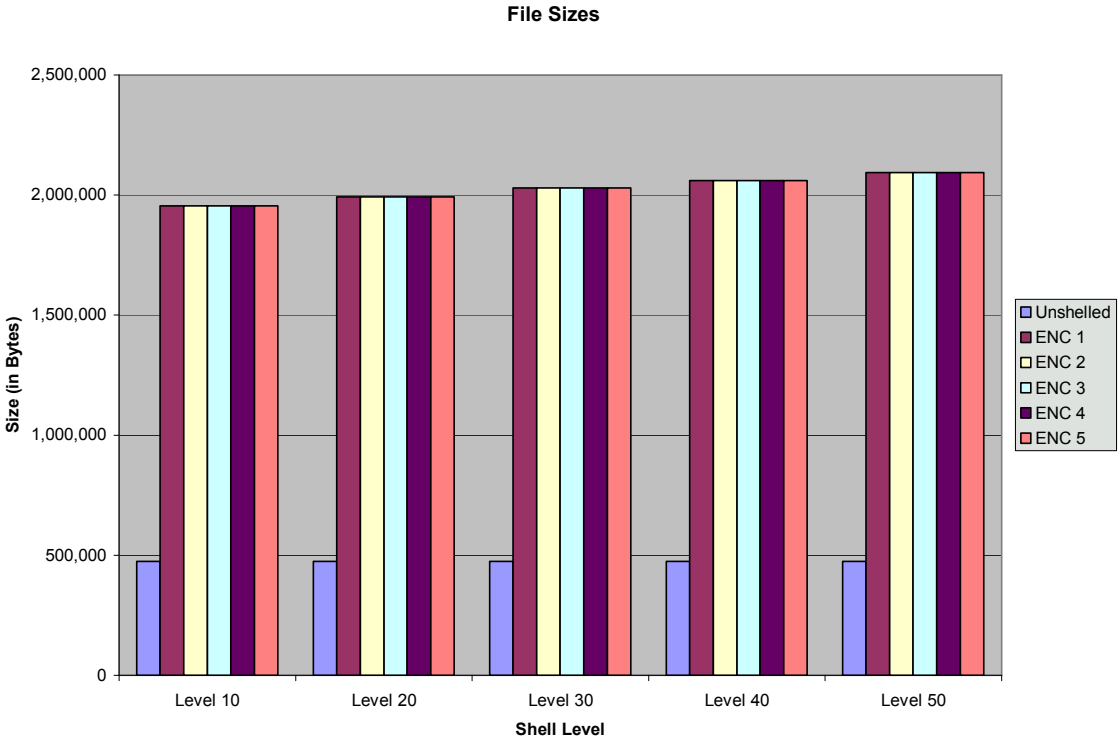


Figure 1.7 – Bar graph of the File Sizes in relation to HASP Shell levels (Lower is better)

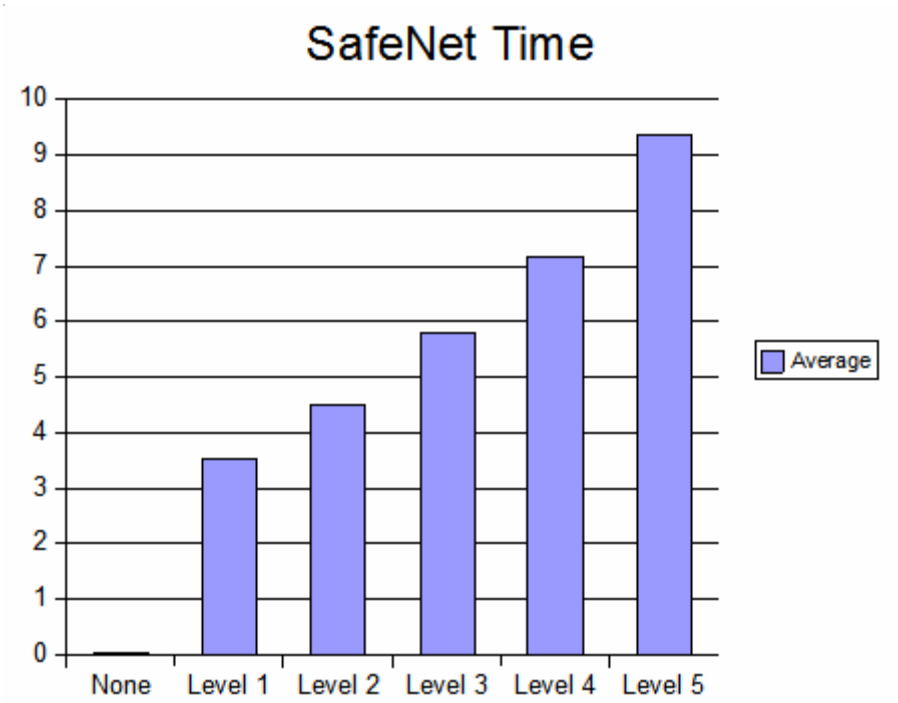


Figure 1.8 – Area graph of Sentinel Time trial in seconds (Lower is better)

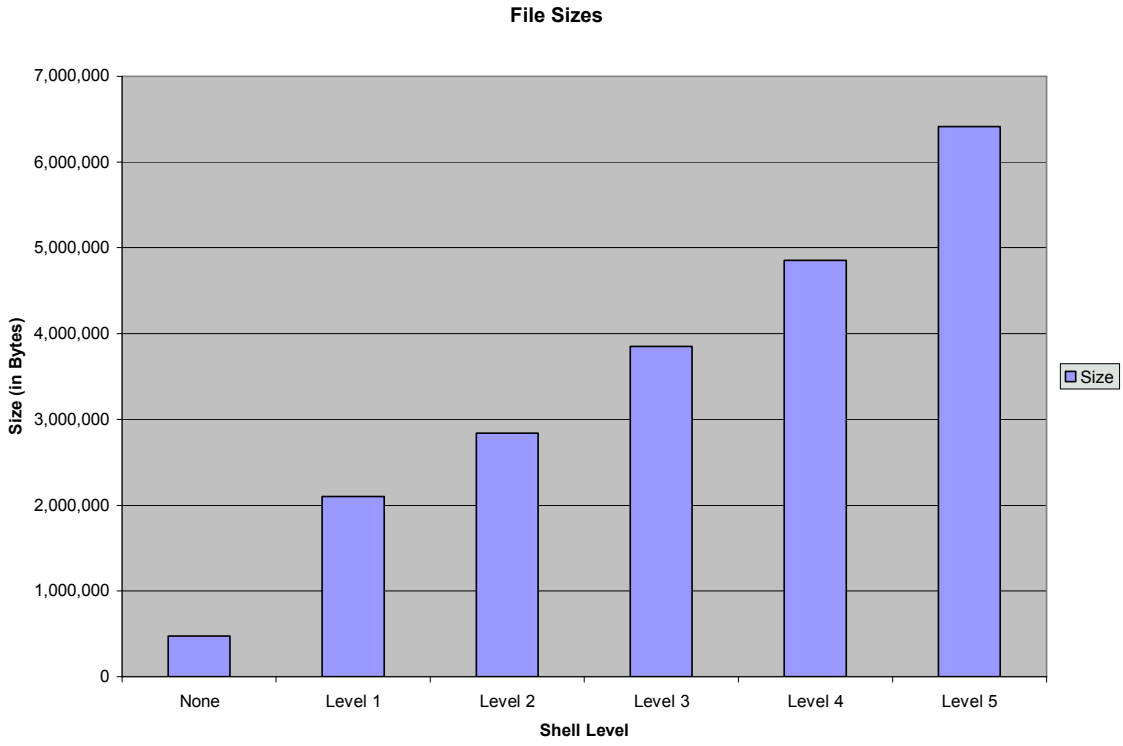


Figure 1.9 – Sentinel File Sizes (Lower is better)

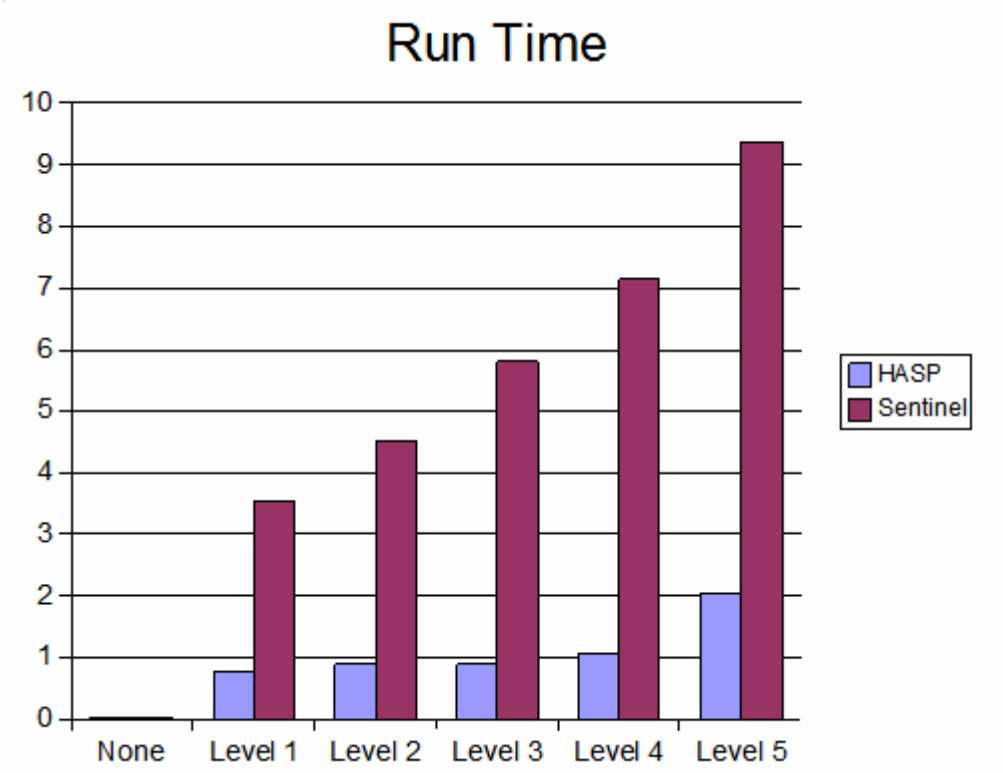


Figure 1.10 – Comparison, SafeNet time vs. Aladdin time, in seconds (Lower is better)

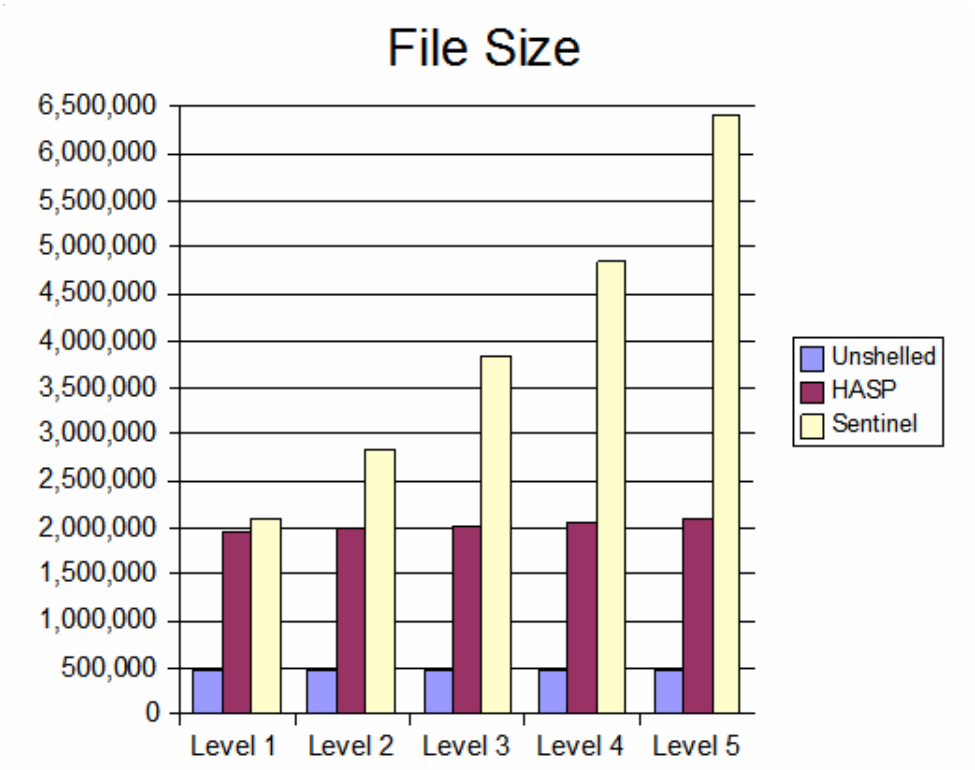


Figure 1.11 – Comparison, SafeNet file size vs. Aladdin file size, in bytes (Lower is better)

Experiment 2: Shell Comparisons

Description

This experiment was designed to discover the foundations of the shelling systems for both the Sentinel and HASP hardware key systems. In addition, a comparison between the two shells was performed to discover the advantages and disadvantages of each.

Experimental Design

Since the actual design of the system is closed-source, the researchers were only able to discover the features of the two different hardware keys by way of company documentation – Aladdin's Programmer's Guide and SafeNet's Developer's Guide documentation.

Data

HASP

- Currently there is a range of protective layers for the HASP key
    - Ranges from 1 to 50 in value
    - Default is 12
- Can also define the number of times a program needs to check against the HASP key
- Can encrypt/decrypt data files as well as executables
- Each layer is encrypted differently than the others
- The more layers used, the harder it is to fully decrypt
- Layers are segment of code, one after another in sequence
- Layers are randomized to increase security

Figure 2.1 – HASP Shell notes

Sentinel

- Has a range of levels for the shell (1 to 5)
- Works by using a multi layering system
- The current layer can only be decrypted if the previous layer was successfully decrypted
- Layers are randomized so no two layer applications are the same (Claim?)
- Shell is able to detect debuggers like SoftICE
- Use the SDK to encrypt program constants, string and code blocks
- Protects against memory dumps
- Uses maze technology and dummy macros
- Uses 128 AES encryption
- Additionally, data files can be encrypted with the shell

Figure 2.2 – Sentinel Shell notes

Results

Based on the documentation provided by both Aladdin and SafeNet, each hardware key utilizes a shell system. A shell is a way to protect a particular application without having to recompile the application again. In both systems, a multi-level shelling architecture is

used. Each layer can only decrypt the next layer, which makes it harder for a hacker to override the system.

As well as both using layers, each shell has added features like debugger detection technologies as well as anti-hacker technology to stop the bypassing of security. In the Sentinel system, features such as dummy macros and Maze technology are used to integrate the shell code into the executable. The HASP system uses false system calls and what the HASP documentation defines as "anti-reverse engineering measures."

Additionally, both shell systems will randomize the layers of each shell to decrease the chance of a hacker correctly breaking or overriding the protection schemes of the software. However, based on the data collected a clear advantage of one system over the other is not apparent.

It would appear that both the HASP and Sentinel keys utilize an equal number of features in their shell system to help deter hackers from correctly decrypting data or gaining unauthorized access to the program.

	HASP	Sentinel
Uses Shell/Envelope	4/4	4/4
Shell has levels	1	1
Levels are randomized	1	1
Anti-debugging features	1	1
Reverse-engineering tech	1	1
Total	100%	100%

Experiment 3: Security Measures

Description

This experiment is designed to discover the security measures performed by both keys to protect data communications between the key and the client machine. In either case, an encryption tunnel is created by both systems, one with the use of ECC and the other with RSA.

Experimental Design

Since the actual design of the system is closed-source, the researchers were only able to discover the features of the two different hardware keys by way of company documentation – Aladdin's Programmer's Guide and SafeNet's Developer's Guide documentation.

Data
HASP

- Data is encrypted/decrypted via AES 128-bit keys, but says nothing about USB traffic
- States that USB traffic/communications is "scrambled"
- Discovers the HASP key by performing system calls to encode/decode data

Figure 3.1 – Notes on the HASP traffic

Sentinel

- Uses 163-bit ECC encryption
- Uses ECDSA & ECSSH to create signatures
- Uses ECKAS-DH1 for the key exchange
- AES packets are in challenge/response format

Figure 3.2 – Notes on the Sentinel traffic

Results

Based on the documentation provided by both Aladdin and SafeNet, the following conclusions can be drawn about the state of protection in regards to USB traffic between the keys and the client machine.

In the HASP documentation, there is little information about the security between the key and the machine to which it is attached. It appears that the key is sent encrypted information, which it will decrypt and send it back to the machine. The HASP documentation states that the traffic is "scrambled," but it does not go into any further detail.

In the Sentinel documentation, there is a detailed explanation of what steps are taken to provide security to the physical traffic between the key and the computer to which it is attached. The sentinel key utilizes an ECC tunnel to send traffic that has already been AES-encrypted. The tunnel is encrypted with a 163-bit ECC key based on the NIST government recommended curves.

Both systems appear to use some sort of protection to secure the information being passed between the key and the client machine. The particulars of this protection are not fully divulged by either party.

	HASP	Sentinel
Data is protected	1	1
Uses encrypted packets	?	1
Uses encrypted tunnel	0	1
Total	1/3 (33.33%)	3/3 (100%)

Experiment 4: Encryption Levels

Description

This experiment is a comparison of the various "encryption levels" that both Aladdin and SafeNet are capable of employing. An encryption level is the strength and security level that the encryption scheme will have. Its effects are speed of execution and time to break into the underlying program.

Experimental design

The design of this experiment is to read the documentation to compare the encryption used by each of the keys. The documentation read for Aladdin's HASP consists of the HASP HL FAQ, the HASP HL Developers' Guide, and the Aladdin website for the HASP HL key. SafeNet's documentation was the SafeNet Sentinel's Developers' Guide, the SafeNet "Curtailing the Piracy Epidemic" report, and the SafeNet's "Sentinel Hardware Keys" document.

Data/Results

The SafeNet Sentinel key and Aladdin's HASP key both utilize 128-bit AES encryption. 128-bit AES is an extremely strong encryption algorithm and, therefore, would be very difficult to break with contemporary equipment.

	HASP	Sentinel
Uses AES 128-bit	1	1
Total	100%	100%

Experiment 5: Tiered Licensing

Description

In this experiment, the tiered system of licensing is compared. The tiered system of licensing refers to the number of levels between customer and developer. Features of each tiered system were examined.

Experimental design

This experiment compares the two tier systems based on the documentation released by the each company. For Aladdin's HASP HL key, the HASP HL Developers' Guide and the HASP HL FAQ were used. For SafeNet's Sentinel key, the Sentinel's Developers' Guide, The SafeNet "Curtailing the Piracy Epidemic" report, and SafeNet's "Sentinel Hardware Keys" document were used.

Data/Results

Comparing the two documents closely, several similarities and differences were noted. First, the SafeNet product uses a three-tiered architecture with the developer at the top. The developer key grants full control over all programs, files, and features. That key allows encryption of programs, creation of all kinds of keys, and remote update of licenses and software.

The second tier for SafeNet is the distributor key. This key grants the bearer the ability to create new user keys and sell ~~him~~ along with copies of the software. The limitations are that the key can only create user keys that the developer has granted him access to create. For example, if the developer has not granted them the ability to create network keys, then the ability to create such a key is unavailable to the distributor.

The third tier for SafeNet is the user level key. There are several types of user keys. Both vendors offer standard user keys, as well as keys with a real time clock (RTC) on-board and keys that are network-based. Network enabled keys allow a system administrator to grant seats to the license based on the number of licenses purchased.

The Aladdin HASP HL key implements a two-tiered architecture. The developer key is again at the top and has the same rights and privileges as the SafeNet key. The key included with the developer's kit has an individual code that allows each company to have one or many developer's keys.

The second tier is the user key, which is the same as SafeNet's key. There are, however, different styles of keys. The most basic key is the standard key, which has no usable storage for program data. There are additional keys that provide on-board memory (which allows more features to be added into a program), expiration timers, and network functionality with various amounts of network users.

The lack of a distributor key, in the researcher's opinion, is not a large disadvantage for Aladdin. The usability of the distributor keys, in the researcher's opinion, is somewhat limited. However, readers of this document should be encouraged to research for themselves and draw their own conclusions.

	HASP	Sentinel
Total	75%	80%

Sentinel licensing system had the distributor key as another option, but makes licensing slightly more complicated. HASP lost approximately 5 points because they did not have a distributor key.

Experiment 6: On-Key Data Storage

Description

This experiment compares the ability to store data on the Aladdin HASP HL key with the SafeNet Sentinel key. This is a comparison based on documentation and the amount of storage space, encryption, and accessibility.

Experimental design

In this experiment, the data storage ability of the keys will be compared based on the documentation. For Aladdin's HASP HL key, the HASP HL Developers' Guide and the HASP HL FAQ were used. For SafeNet's Sentinel key, the Sentinel's Developers' Guide,

The SafeNet “Curtailing the Piracy Epidemic” report, and SafeNet’s “Sentinel Hardware Keys” document were used.

Data

Aladdin's HASP HL key options are quite extensive concerning memory options. The lowest end key with memory that Aladdin's set comes with is the HASP M1 key. This key comes with 112 bytes of on board memory for storage of data, which is encrypted for protection. This allows very little to be saved to the key besides a very short string. The other keys that Aladdin makes, however, all come with 496 bytes of on board memory.

The SafeNet keys, on the other hand, have 8k of on board memory for storage of developer data. This allows interactivity between the key and the software to ensure that the program will not run without the key present. For example, a string could be saved to the Sentinel key that would be required in order to open a function, or continue past a random checkpoint that the developer could choose to put in the code. This would ensure that without the key present, the software would not be able to run.

	HASP	Sentinel
Memory	1/9	9/9
>= .25 KB	1	1
>= 1 KB	0	1
>= 2 KB	0	1
>= 3 KB	0	1
>= 4 KB	0	1
>= 5 KB	0	1
>= 6 KB	0	1
>= 7 KB	0	1
>= 8 KB	0	1
Total	11.11%	100%

Analysis

Based on the data retrieved by experimentation as well as the documentation for both products, the following conclusions can be drawn:

- Both systems utilize a multi-layer system for shelling;
- Both systems have an increase in file size as the shell levels increase
  (Sentinel file sizes are larger overall than their HASP counterparts);
- Both systems have an increase in program load latency as the shell levels increase
  (HASP load time appears to be less than the time for Sentinel protected executables);
- Both systems have some form of protection in regard to the communication between the key and system
  (HASP states the protection to be only that communications are “scrambled” while Sentinel documentation provides an overview of the encryption schemes used in the tunnel and encrypted data packets);
- Both systems have multiple keys for different styles of licensing

(HASP is a little easier to understand while Sentinel is a little more robust and flexible).

Summary

Both products offer many of the same features and protection styles in slightly different ways. Having studied the way that these tools are protecting software, both appear to have their strengths and weaknesses. SafeNet's Sentinel product is more flexible with one or two options. Aladdin's HASP product seems a little easier to understand. Ultimately, both products do very well at securing the software they are attempting to secure. The increase in file size and execution time for SafeNet is an indication that its shell might be doing something more in the way of protection than Aladdin HASP.

Scoring

Protection Architecture	**Aladdin** HASP HL	**SafeNet** Sentinel Hardware Key
Shell Execution Times	84.62	27.69
Shell Comparisons	100	100
Security Measures	33	100
Encryption Levels	100	100
Tiered Licensing	75	80
On-Key Data Storage	11	100
Overall	**54.97**	**71.42**

This scoring ultimately reflects the ability of each product to secure an application quickly and efficiently. Scores were tabulated based on percentages of points out of 100%. In the opinion of the researchers, SafeNet Sentinel earned a higher score because its most basic shell is more integrated into the protected application than Aladdin's envelope. This is not a bad thing for Aladdin's product overall, as it relies more heavily on the use of API calls to protect an application. Both the shell and the API should be used to completely protect an application, but due to time and technical difficulties, the research team was unable to test using an application protected with the full feature set of the API. This is something that should also be considered for any potential customer of these products.

References

Aladdin Inc. HASP HL - Features & Benefits. http://www.aladdin.com/HASP/features_benefits.asp. Aladdin Knowledge Systems Ltd. 2006.

Aladdin Inc. HASP Programmers Guide v1.3 Aladdin Knowledge Systems Ltd. 1985.

Aladdin Inc. HASP HL F.A.Q. Aladdin Knowledge Systems LTD. 1985.

SafeNet Inc. SafeNet Sentinel Developer's Guide. SafeNet Inc. Baltimore. 2005.

SafeNet Inc. Curtailing the Piracy Epidemic: A Case for Hardware Security Keys. SafeNet Inc. Baltimore. 2005.

Section 4: Remote Updates

Introduction

In this phase, tests will be performed to measure the ability of the USB keys to perform a remote update using the encrypted files sent over e-mail or CD. Remote licensing updates for USB keys rely heavily on this method. As a result, this is a key feature for any company wishing to add execution counts, add users or features, or remove the licensing requirement altogether.

Features

Both USB keys have a utility included in the driver install that allows for remote update. All that is needed is to execute the utility with the user key in the USB port. The program generates a file, which is sent to the developer either via email, or burned to a CD and sent via the mail service. Its size is small enough, however, to allow for electronic transfer without worrying about bandwidth or quota issues. The distributor will then use this file to generate whatever licensing change is pertinent to the request. The licensing change will then be sent back to the user in a similar file. The size of the file is not a factor with the update. Upon receipt, the user will update the license using the update utility tool. The user imports the file and the license updater will update the key with the new licensing information.

Both manufacturers provide remote update broadcast capability, which can allow for updating in one location via an internal network. This update is propagated to all the users on that corporate network. This applies to the usability of the product from the user perspective. It allows a system administrator implementing a license update to do the job once instead of multiple times. However, other factors, such as the bandwidth of the network, number of hosts to be updated, and network performance need to be considered.

Experiments that were performed

The experiments that were performed tested the ability of the keys to update remotely without the need for a distributor or developer to come to the location and manually update the license on each computer. Tests measured the ability to update one machine's license request for a license increment. The researchers also attempted to update one machine's license requests with more complex changes (such as additional features, users, and execution counts) and to update license changes over a broadcast server.

Interpretation of Results

The results will be interpreted based on usability, functionality, feature sets, and security. All will be given the same weight, however, if one area is so poor that it makes the feature or product unusable, the lack of functionality may affect scores in other areas.

Methods of Update

SafeNet

Directions:
Once the executable was created, the SafeNet toolkit needed to be run. In the license designer, a new wizard was opened via the "open wizard" option. The user would then start the "new to creating features and templates" and then chose the "continue" option. After the user added their executable, they then entered their destination path. On the next screen, they chose the "limit executions" option. The user would then set the execution count to one and select "add instances later." After two clicks on the "continue" option, a license template name was entered, and the finish button was selected.

With both keys in the USB ports, the user then selected the user key from the "USB status" menu. The "select build" option was utilized. Then, the user selected the "make key" option to make a user key to protect the application with a shell. A test of the execution limitation was advisable here to make sure that the program could only be run once.

In order for a user to continue running the application, an update was needed to modify their license. The user would then run the secure update utility and select the "generate request code" option. Next, open up the toolkit again and go to the "update manager" tab. An action of type "license / feature" would be added and the program name that is being updated would be selected. On the right window, under the "commands" drop down window, the "Increment execution counter" was selected. The value was set to one, and after the action was named. The "OK" button was clicked.

The vendor then selected the "Key activator" tab, followed by the "load request code" button. After selecting the file created with the update utility, the vendor would then select the "execution count updater" action that was just created. The "Generate update code" option was selected and saved to a file.

The client received the file, opened the update utility again, and selected the folder button. The client then browsed to the location of the file and chose the "OK" option. The client then selects the "activate application" button, which will perform the update. After the application was tested again, the researchers were able to verify their update.

HASP

In order to update the HASP key, the user must run a custom-created HASP Remote Update executable. The vendor must create a remote update service (RUS) for each encrypted application with a customized message. Once the user creates their file, they can distribute that file to the vendor via their preferred mechanism. The vendor can then open the file and process an order to change the key based upon the specifications of the vendor. In this experiment, the key was changed from three executions to an unlimited key. Once the vendor to client file is created, it can then be redistributed to the client and the update applied.

Experiment 1: Transferring License Flat Files

Description

This experiment is a basic test of updating the license with flat files sent over e-mail or USB with attention being paid to testing time involved and difficulty.

Topology and experimental design

This topology includes two computers both isolated from the network. The first computer, PC1 was the developer machine. This machine created the user key and application to be encrypted. It used a simple shelled application created with the wizard. It was created with an execution count of one. The second computer, PC2 had this application installed on it and was run once to exhaust the execution count. The program was then run again to confirm the program would not run. The user on PC2 then used the toolkit included to request an updated license. The file was then transferred to PC1 using a USB thumb drive. The developer on PC1 then created an update file, which would increment the execution count by one upon receipt. This update file was then transferred back to PC2 using the USB thumb drive. The user at PC2 then updated the license using this file and the toolkit. The program was then executed twice again on PC2 to test the update.

Data/Results

From the experiment, we have determined that as long as the files are intact, the process will succeed. The method of moving the update files between the client and the developer does not affect the results of the process and works equally well for Aladdin and for SafeNet. This method of doing updates is unfortunately slow, and requires the developer to create the update file. The files sent are, for Aladdin, C2V, and V2C files and for SafeNet, REQ, and UPW files. Both files are extremely small (less than 1kb) so with the number of licenses of software sold; these could be archived extremely easily without much cost.

Evaluation

HASP	90 out of 100
SafeNet	85 out of 100

The Aladdin and SafeNet keys both completed this experiment and did not have any problems with the update processes. However, the process was a little slow due to the mailing of files, so it lost points there. To the experimenters, the licensing model for SafeNet seemed more complicated to set up for updating the products used. The distributor key was not taken into account in the scoring because it could not perform remote updates. The experimenters did remove points because they were unable to find documentation that clearly stated that the distributor key could not be used for remote updates.

Experiment 2: Complex License Flat File Transfer

Description

This experiment was intended to perform a more complex test of updating the license with flat files sent over e-mail or USB, with attention being paid to testing time involved and difficulty. However, due to time constraints and technical difficulties the proposed experiment was not performed.

Topology and experimental design

This topology included two computers both isolated from the network. The first computer, PC1 was the developer machine. This machine created the user key and application that was encrypted. It used a simple shelled application created with the wizard and has basic API functionality built in. It was created with an execution count of one and no access to a certain function or application. The second computer, PC2 then had this application installed on it. The application's features were both tested to ensure only one worked. This first execution exhausted the execution count, and the application was executed again to confirm this. The user on PC2 then used the toolkit included to request for an updated license. The file was transferred to PC1 using a USB thumb drive. The developer on PC1 then created an update file, which incremented the execution count by one and granted access to the protected function. This update file was then transferred back to PC2 using the USB thumb drive. The user at PC2 then updated his license using this file and the toolkit. The program was then executed and both functions were tested. The program was then executed again to ensure the execution could expire when it should.

Data/Results

The tests could not be completed due to technical difficulties and time constraints. Ultimately, this would have been successful for both applications based off the manufacturers' documentation and an examination of the user interface.

Evaluation

Aladdin	100 out of 100
SafeNet	100 out of 100

This test could not be performed because the experimenters failed to create working, multi-featured applications that utilized the API. The experimenters did examine the interfaces and the documentation of both products to examine how easy this would be to perform. Both appeared to be reasonably simple and the documentation appeared clear. Because the test was not performed and no flaws were found at a cursory glance at the interface, it was decided neither company would have points deducted. Any developer wishing to allow complex licensing updates, such as those outlined in the experiments, is encouraged to test these features.

Experiment 3: License Update to Several Clients

Description

This experiment was intended to be a more complex test of updating the license with flat

files sent over email or USB and with several network clients with a single server. The experiment would have been performed with attention being paid to testing time involved and difficulty. However, due to lack of resources and time, the testing was not actually performed.

Topology and experimental design

This topology included four computers with one isolated from the network. The other three computers, PC2, PC3, and PC4, were networked together. The first computer, PC1 was the developer machine. This machine created the user key and application that was encrypted. It used a simple shelled application created with the wizard and had basic API functionality built in. It was created with an execution count of three and no access to a certain function. The second computer, PC2 then had this application installed on it and setup as a central server. PC3 and PC4 were then pointed at PC2 looking for the license to run the program. All functions were tested on each machine to ensure that only one would run. The program was then run again to confirm that the license expired after three executions. The user on PC2 then used the toolkit included to request an updated license. The file was then transferred to PC1 using a USB thumb drive. The developer on PC1 then created an update file, which incremented the execution count by three and granted access to the protected function. This update file was then transferred back to PC2 using the USB thumb drive. The user at PC2 then updated his license using this file and the toolkit. All functions and features were then executed again to ensure that the previously protected function executed. Finally, the application was then run a fourth time to ensure that the execution update worked correctly.

Data/Results

Due to lack of resources, the experimenters were unable to test this functionality. However, data was collected based on documentation and an examination of the user interfaces. Both indicated that this would work correctly and completely and both appeared to provide the ability to update network licenses remotely.

Evaluation

HASP	90 out of 100
SafeNet	85 out of 100

Both Aladdin and SafeNet documentation stated that updating network licensing would be as simple as updating standalone keys. Based on this, and the user interfaces available, it was decided that the scores for this experiment would be identical to that of experiment 1. The reasoning for this is because everything is the same, and both appear to work identically. Without any testing to award additional, (or subtract) points from either company, these are the scores reached. If this feature is important for one's application, the experiment could be performed following the procedure outlined above using the network enabled user keys for both companies.

Analysis

Experience with both companies' products was mixed. Concerning SafeNet, the

experimenters encountered difficulties understanding the use of the licensing structure. The additional keys provided an extra concept for the researchers to understand. This made it hard for the researchers to understand fully the uses of all the various keys. The primary example of this was the expectation (of the experimenters) that the distributor key would be able to perform remote updates. This was not the case. Upon further reading of the documentation, it was understood that the key did not actually work in this fashion. To the researchers, however, this could have been clearer.

The documentation for SafeNet was clear about the way remote update worked. This greatly helped in the understanding of how remote update procedures worked conceptually. The user interface for SafeNet provided a wizard, which allowed for the easy creation of licenses, but certain options to allow for updating of licenses were not enabled by default. This required more research through the documentation to discover which options were needed, and which were not. Fortunately, the information for what each option did was readily available in the documentation. The remote update tool included with SafeNet's product can be left as a default tool located in the same directory as the application being protected. It can also be customized using the API functionality.

Concerning Aladdin, their documentation was equally clear with regard to the conceptualization of Remote Update. Their documentation was also clear on the way that one could update an application. However, Aladdin's update application required customization for each application created. Aladdin's tool does, however, create a file almost identically sized and can be written using the API.

The interface to create an application that could be updated was relatively straightforward and included a wizard that allowed for the easy creation of such an application. The Aladdin tool requires a number of options to be changed in order for remote updating to be possible. However, this is all clearly documented in the application. The Aladdin licensing structure uses two styles of keys, which allows for very simple understanding of what each does.

Summary

Both companies are striving to create high quality products. Aladdin's remote update and SafeNet remote update both implement the remote update procedure in similar ways. Both require that the developer ship a license update utility with the product and the keys. The documentation is adequate for both companies. However, Aladdin's was slightly more in-depth with how to configure and use the remote update tools. SafeNet relies more on the wizards than in-depth documentation, which is not a bad thing, but if a user has a problem using the wizard, then in-depth documentation would be useful. SafeNet's concept of the distributor key is also a big step in allowing the distribution of a product to be easier, which could allow for faster delivery of a protected application. However, because this key could not create remote update files, it was not applicable in this section of the report.

Scoring

Remote Updates	**Aladdin** HASP HL	**SafeNet** Sentinel Hardware Key
Transferring License Flat Files	90	85
Complex License Flat File Tx	100	100
License Update to Several Clients	90	85
Overall	**93**	**90**

The scoring ultimately reflects the difference between the ease of use of both products. SafeNet's product has a default application that will perform the remote update procedures. They also allow the same level of customization as Aladdin. However, while Aladdin's tool requires customization before it will work correctly, it is well documented on how to do this, and includes a wizard to step you through the procedure. Aladdin's developer licensing tool for creating the update file is, in the opinion of the experimenters, easier to use. Both companies lost points for not allowing an intermediary party, such as a distributor, access to sell or create remote license updates. In addition, both companies lost points for not implementing the ability to allow a client to purchase remote updates without the need to email files.

References

Vendor Documentation
http://www.aladdin.com
http://www.safenet-inc.com

Section 5: Drivers and Libraries

Introduction

This section attempts to evaluate hardware token authentication on the merits of the packaged device drivers. The functionality testing is limited to Microsoft Windows XP with Service Pack 2, but does explore the option of additional supported operating systems.

For this section, there are 100 total points available. Each experiment was equally weighted and the scores were averaged. A passing score of 70 was awarded for experiments that met expectations. Additional points, up to total, were awarded for exceeding expectations.

Experiment 1: Driver Availability

Which operating systems are supported by drivers available from web resources such as the vendor website, Windows Update, or open-source projects?

Assertion

Although the scope of comparison in most tests conducted in this report was limited to Microsoft Windows XP SP2, it would be remiss to exclude other supported platforms. The Aladdin HASP HL offered support on a greater number of platforms, and had a more intuitive path to the drivers on the manufacturer's website. Drivers were available for Windows, Mac OS X, and two major Linux distributions, in addition to scripted command line installs.

SafeNet's drivers were not difficult to find, but were limited to only 32-bit versions of Microsoft Windows. SafeNet's online text is still listing "Rainbow Technologies" in several locations (Figure 1.3), despite completing a merger with the company in March of 2004 (Figure 1.4).

HASP-HL 100
Sentinel 70

Experimental Design

Following the included and online documentation, the researchers verified that all drivers were available for installation. Successful driver installation in Windows is achieved when the device is displayed properly in Windows Device manager.

Analysis

Aladdin Steps

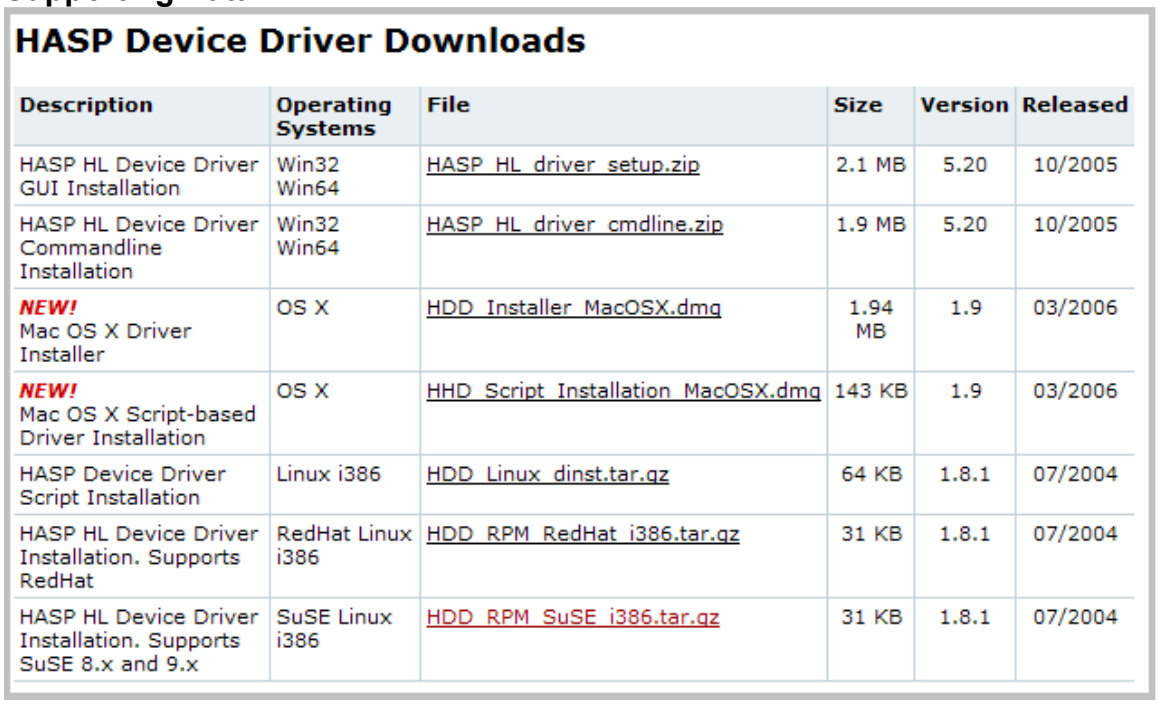
1) Navigate to www.aladdin.com
2) Click "Support & Downloads" from the top banner bar
3) Choose "HASP" from the drop down menu
4) Click "End Users" below the HASP HL heading

5) Scroll to the appropriate operating system driver download link. Some scrolling required at resolutions 1024 x 768 and below (Figure 1.1).

SafeNet Steps

1) Navigate to www.safenet-inc.com
2) Hover over "Technical Support" from the top banner bar
3) If JavaScript is enabled, choose "Basic Support" from the pop-up menu
4) Click "Sentinel Hardware Key Downloads/Help Tips (Ultrapro, Superpro, Pro)" from the "Top Downloads/Help Tips" section
5) Click on "Sentinel Hardware Key". If JavaScript is enabled, driver install instructions will appear.
6) Choose "protection" or "driver only" installation type, and download the appropriate link. Some scrolling required at resolutions 1024 x 768 and below (Figure 1.2).

Supporting Data



HASP Device Driver Downloads

Description	Operating Systems	File	Size	Version	Released
HASP HL Device Driver GUI Installation	Win32 Win64	HASP_HL_driver_setup.zip	2.1 MB	5.20	10/2005
HASP HL Device Driver Commandline Installation	Win32 Win64	HASP_HL_driver_cmdline.zip	1.9 MB	5.20	10/2005
NEW! Mac OS X Driver Installer	OS X	HDD_Installer_MacOSX.dmg	1.94 MB	1.9	03/2006
NEW! Mac OS X Script-based Driver Installation	OS X	HHD_Script_Installation_MacOSX.dmg	143 KB	1.9	03/2006
HASP Device Driver Script Installation	Linux i386	HDD_Linux_dinst.tar.gz	64 KB	1.8.1	07/2004
HASP HL Device Driver Installation. Supports RedHat	RedHat Linux i386	HDD_RPM_RedHat_i386.tar.gz	31 KB	1.8.1	07/2004
HASP HL Device Driver Installation. Supports SuSE 8.x and 9.x	SuSE Linux i386	HDD_RPM_SuSE_i386.tar.gz	31 KB	1.8.1	07/2004

Figure 1.1 – HASP driver download page

Step 1 - Driver Update

Many technical problems involving our Sentinel keys can be resolved simply by installing the latest system drivers. Please download and install the latest driver below.

Sentinel Keys Protection Installer v1.0 - (Download Driver) *For Windows 98/ME/NT/2000/XP/Server 2003 (Server only on 2000/XP/Server 2003)*

This is the Sentinel Keys Protection Installer. This installer will install both the driver and server file for the Sentinel Hardware Key product. Please note, this is not the driver for the Sentinel SuperPro or Sentinel Ultrapro. Please go to the Sentinel SuperPro or Ultrapro sections for their respective drivers.

This will work for the following key: Sentinel Hardware Key

Sentinel Keys Driver Installer v1.0 - (Download driver)
For Windows 98/ME/NT/2000/XP/Server 2003

This installer will install the Sentinel Keys Driver only. Please note, this is not the driver for the Sentinel SuperPro or Sentinel UltraPro. Please go to the Sentinel SuperPro or UltraPro sections for their respective drivers.

Figure 1.2 – SafeNet driver download page

Disclaimer

Please note, that though we will make the best effort to help you, we are limited on how much support we can actually provide. For most problems, intimate knowledge of the software package's security is needed. This is information that Rainbow Technologies is not privy to. For this reason, it is always a good idea to **contact the developer of your software product** for support. We are limited to the help only with:

Copied from:
http://www.safenet-inc.com/support/tech/sentinel.asp

- driver installation issues
- minor configuration questions

Figure 1.3 – "Rainbow Technologies" listed in Sentinel Hardware Keys support information

What you should know about SafeNet

SafeNet, Inc. (Nasdaq: SFNT), a global leader in information security, offers an integrated suite of encryption products—hardware, software, intellectual property, and chips— protecting communications, business data, and digital identities.

In 1983, the company, started in a basement in Timonium, MD by two NSA engineers, was born as Information Resource Engineering (IRE). It specialized in selling enterprise network security solutions, using encryption technology to protect the public and private networks of financial institutions. Soon it expanded into the federal government sector.

IRE's first investor was Tony Caputo, current SafeNet Chairman, and CEO. By the time the company was renamed SafeNet, Inc. in 2000—taking the name of its award-winning VPN product line— it had already embarked on a path that would make it one of the world's largest and most respected security organizations.

The strategic acquisitions of Securealink BV in the Netherlands in 2002, and Cylink Corporation in 2003, keyed rapid growth. When SafeNet merged with Rainbow Technologies in March 2004, it became the world's seventh largest information security company. The merger more than tripled SafeNet's revenue base and effectively doubled the number of shares outstanding.

Figure 1.4 – SafeNet merger completion - http://www.safenet-inc.com/company/history.asp

Experiment 2: Driver Installation (Windows XP)

What steps are required to install the hardware token on Windows XP SP2 interactively?

Assertion

The easiest method for installing drivers is the command line method supported by HASP HL. It is suggested by both companies to include the drivers within the product's installer. All methods listed below resulted in a successful driver installation. There is an alternate installer on the SafeNet website that does not include the extra Sentinel Server.

To remove the drivers:
HASP HL: %CD%:\Windows\Installed\Drivers, run haspdinst.exe -r
Sentinel Hardware Key: Re-run the driver installer and choose remove.
Both utilities remove the drivers completely.

HASP-HL 100
Sentinel 90

Experimental Design

Following the instructions included with the development kits, list the steps required to complete driver install. A successful driver installation is measured by the device recognition in Windows Device Manager.

Analysis

HASP

The HASP HL documentation (HASP HL Software Protection and Licensing Guide) lists 2 supported methods of software installation: HASPUserSetup.exe and haspdinst.exe. The HASPUserSetup.exe is a GUI-driven installation program designed for Windows98 through Windows Server 2003.

1) Insert the HASP HL CD-ROM.
2) Choose "Browse the HASP HL CD."
3) Run the installation utility (from the path %CD%:\Windows\Installed\Redistributable\Drivers\Setup, run HASPUserSetup.exe).
4) Click Next.
5) Accept the license agreement and then click Install.
6) Click Finish.
7) Reboot.

Alternate Method from the command-line:

1) Insert the HASP HL CD-ROM.
2) Choose "Browse the HASP HL CD."
3) Run the installation utility (from the path %CD%:\Windows\Installed\Drivers, run haspdinst.exe –i).
4) When the installer is complete, it will display a message.

The following files are installed:

%windir%\system32\drivers\akshasp.sys
%windir%\system32\drivers\hardlock.sys
%windir%\system32 \akscoinst.dll
%windir%\system32\drivers\aksclass.sys
%windir%\system32\drivers\aksusb.sys

Sentinel

The Sentinel Hardware Keys documentation (System Administrators Guide) recommends downloading the driver from the website at http://www.safenet-inc.com/support/index.asp.

1) Launch your web browser.
2) Navigate to http://www.safenet-inc.com/support/index.asp.
3) Click on Sentinel Hardware Key Downloads/Help Tips (UltraPro, SuperPro, Pro).
4) Click on Sentinel Hardware Key
5) Click on download driver next to the heading **Sentinel Keys Protection Installer v1.0. (**The file is 5.3 MB in size.)
6) Double-click on the file **Sentinel Keys Protection Installer 1.0.0 (English).exe**
7) Click Next.
8) Accept the license agreement and then click Next.
9) Choose **Complete** installation and then click Next.
10) Click Install.
11) Click Yes (default) to allow the installer to modify your firewall settings (A command line window will appear.)
12) Click Finish.

The following files are installed:

%windir%\system32\drivers\skeysusb.sys

In C:\Program Files\Common Files\SafeNet Sentinel\Sentinel Keys Driver\

skeysusb.cat
skeysusb.inf
skeysusb.sys

For the Sentinel Keys Server, the following additional drivers are installed:

In C:\Program Files\Common Files\SafeNet Sentinel\Sentinel Keys Server\

libeay32.dll
MD5CHAP.dll
PwdGenUtility.exe
sntlconfigsrvr.xml
sntlkeyssrvr.exe

In C:\Program Files\Common Files\SafeNet Sentinel\Sentinel Keys Server\ root

Cancelinfo.html
CancelInfoApplet.class

CancelLicenseRow.class
CMd5Chap.class
CMessage.class
CPasswordDlg.class
keyinfo.html
KeyInfoApplet.class
KeysRow.class
LabelAdapter.class
licenseinfo.html
LicenseInfoApplet.class
LicenseInfoRow.class
LicenseRow.class
licenseUsages.html
LicenseUsagesApplet.class
LicLabelAdapter.class
XMLParser.class

In C:\Program Files\Common Files\SafeNet Sentinel\Sentinel Keys Server\root\resources
AppletResources.class
AppletResources_en.class

Experiment 3: Microsoft Update

Can the drivers be updated automatically via resources such as Microsoft Update?

Assertion

Aladdin's HASP HL drivers are available from the Microsoft Update site. This is noted in their documentation. SafeNet Sentinel Hardware Key drivers are not available.

HASP-HL 100
Sentinel 0

Experimental Design

Since traffic during Windows Update is SSL encrypted, the experiment analyzed windowsupdate.log file stored in the %windir% folder. For this experiment, the user will insert the hardware token on a system. The user selected "Automatically search and install a driver," allowing the system to query the Microsoft Update site over the Internet. The log file was analyzed to verify the response from the website.

Analysis

The Sentinel key was not able to locate an appropriate driver. It was recognized (Figure 3.1) as a USB device, but a search for a matching driver online failed (Figure 3.2). The HASP HL key was also recognized as a USB device (Figure 3.3), but was able to successfully find a driver (Figure 3.4).

Supporting Data

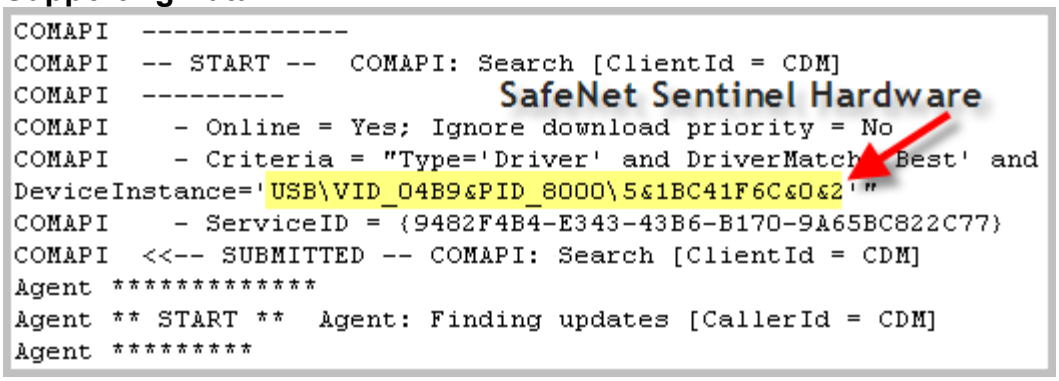

COMAPI -------------
COMAPI -- START -- COMAPI: Search [ClientId = CDM]
COMAPI ---------
SafeNet Sentinel Hardware
COMAPI - Online = Yes; Ignore download priority = No
COMAPI - Criteria = "Type='Driver' and DriverMatch 'Best' and
DeviceInstance='USB\VID_04B9&PID_8000\5&1BC41F6C&0&2'"
COMAPI - ServiceID = {9482F4B4-E343-43B6-B170-9A65BC822C77}
COMAPI <<-- SUBMITTED -- COMAPI: Search [ClientId = CDM]
Agent **********
Agent ** START ** Agent: Finding updates [CallerId = CDM]
Agent *********

Figure 3.1 – Sentinel key inserted

```
Agent *********
Agent **  END  **  Agent: Finding updates [CallerId = CDM]
Agent *************
COMAPI  >>--  RESUMED  -- COMAPI: Search [ClientId = CDM]
COMAPI    - Updates found = 0
COMAPI  ---------
COMAPI  --  END  --  COMAPI: Search [ClientId = CDM]
COMAPI  -------------
CDM WARNING: CCdm::ExecuteSearchForOneDriverUpdate failed, error = 0x80240024
CDM WARNING: CCdm::FindMatchingDriver failed, error = 0x80240024
CDM WARNING: FindMatchingDriver failed, error = 0x80240024
Report  REPORT EVENT: {74C25D04-04CB-45D7-95EA-445F06AA693C}  2006-05-15
07:12:50-0400 1 147 101 {00000000-0000-0000-0000-000000000000}  0 0 CDM
Success Software Synchronization  Agent has finished detecting items.
CDM CancelCDMOperation
```

Figure 3.2 – No driver found

```
COMAPI  -------------
COMAPI  -- START --  COMAPI: Search [ClientId = CDM]
COMAPI  ---------
COMAPI    - Online = Yes; Ignore download priority = No
COMAPI    - Criteria = "Type='Driver' and DriveMatch='Best' and
DeviceInstance='USB\VID_0529&PID_0001\5&1BC41F6C&0&1'"
COMAPI    - ServiceID = {9482F4B4-E343-43B6-B170-9A65BC822C77}
COMAPI  <<-- SUBMITTED -- COMAPI: Search [ClientId = CDM]
Agent *************
Agent ** START **  Agent: Finding updates [CallerId = CDM]
Agent *********
```

Aladdin HASP HL

Figure 3.3 – HASP HL key inserted

```
DnldMgr *************
DnldMgr ** START **  DnldMgr: Downloading updates [CallerId = CDM]
DnldMgr *********
DnldMgr   * Priority = 3, ServiceId = {9482F4B4-E343-43B6-B170-9A65BC822C77}
DnldMgr   * Updates to download = 1
Agent   *   Title = Aladdin Knowledge Systems LTD. - Other Hardware - Aladdin
USB Key
Agent   *   UpdateId = {5E9CB552-709A-4A15-BA13-530A84E2D36A}.100
```

Figure 3.4 – Driver found and installed

Experiment 4: Driver Certification

Are the drivers certified for use on Windows XP SP2?

Assertion

The Aladdin HASP HL Basic, Pro and Net are certified for use with Microsoft Windows XP Home Edition and Microsoft Windows XP Professional Edition. A SafeNet Sentinel driver is mentioned as well, but not as specifically as in Aladdin HASP products. It is not clear that the driver listed in Windows Hardware Quality Labs (WHQL) is the driver packaged with the product. As the research showed, it is not the same driver as the one included or the one available online, which are not WHQL certified (Figure 5.1).

HASP-HL 100
Sentinel 0

Experimental Design

Research the components using the Microsoft Windows Marketplace.

Analysis

The following HASP products were listed: HASP HL Basic, HASP HL Pro, HASP HL Max, HASP HL Time, HASP HL Net, HASP HL NeTime, HASP4 USB M1, HASP4 USB M4, HASP4 USB Standard, HASP4 USB Net, HASP4 USB Time, and Hardlock USB.

The following SafeNet Products were listed, including products previously released by Rainbow Technologies: Sentinel SuperPro and Sentinel USB Security Device Driver

Supporting Data

HASP HL testing results:
http://testedproducts.windowsmarketplace.com/item.aspx?idItem=649fb22d-4e53-ec06-6934-906897526753

Sentinel USB Security Device Driver
http://testedproducts.windowsmarketplace.com/item.aspx?idItem=09110318-3c63-984c-b03a-4e18d0e52ebb

Experiment 5: Driver Redistribution

What steps are required to package the Microsoft Windows XP SP2 drivers with a software project? Can they be incorporated into the product installer?

Assertion

Both packages provided adequate .msn patch files to package the drivers successfully. Upon installation on a new PC image, only the HASP installer successfully recognized and installed the drivers. Sentinel key drivers were copied but required manual installation.

HASP-HL 100
Sentinel 70

Experimental Design

Using Microsoft Visual Studio .NET 2003, the reviewers created a new setup project for an existing solution file and follow the documentation to create an installation package that included drivers.

Next, the reviewers inserted the hardware token into a system that has never had drivers introduced. This allowed the hardware wizard to detect and attempt to automatically install the drivers, after ensuring that the system did not have Internet access. When the automatic install failed to locate an appropriate driver, the team launched the setup.msi file created in the previous step. The team then verified that the hardware token was recognized in the device manager without a "?".

Analysis

The SafeNet documentation (p 158, Sentinel Keys Developer's Guide) explained how and when to deploy the Sentinel drivers, but it did not actually install the driver for use during product installation. It stored the drivers at the path Program Files\Common Files\SafeNet Sentinel\Sentinel Keys Driver. This performed as documented. Windows could not locate the drivers by default at that location. When that path was explicitly chosen, the drivers were then installed properly. The user is prompted during installation with a warning (Figure 5.1) that the software has not passed Windows Logo testing.

Supporting Data

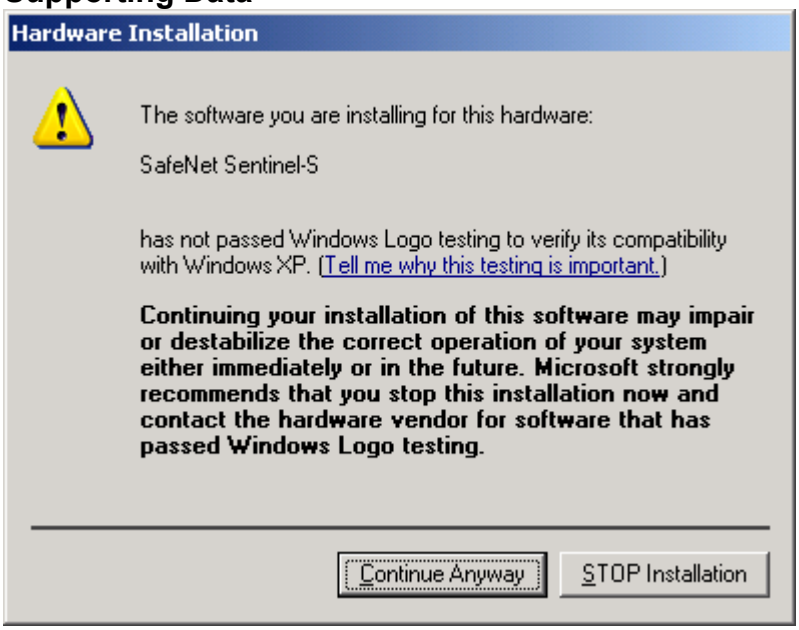

Hardware Installation
The software you are installing for this hardware:
SafeNet Sentinel-S
has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)
Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.
Continue Anyway
STOP Installation

Figure 5.1 – Sentinel Driver not Windows XP verified

Scoring

Drivers & Libraries	**Aladdin** HASP HL	**SafeNet** Sentinel Hardware Key
Driver Availability	100	70
Driver Installation	100	90
Microsoft Update	100	0
Driver Certification	100	0
Driver Redistribution	100	70
Overall	**100**	**46**

The two products reviewed earned widely different scores based mostly on the implementation of driver distribution. While the SafeNet Sentinel provided all of the requisite drivers for the reference platform, the Aladdin HASP HL offered a greater opportunity for developers wishing to create applications on systems other than Microsoft Windows XP. The installation applications for both product performed satisfactorily, but the Aladdin HASP HL offered a command line mode for silent installation and removal. The SafeNet Sentinel installers, while packaged on the CD, encouraged the user to instead download them from the company's website.

This comparison would be much closer if SafeNet were able to obtain driver certification through Microsoft Windows Hardware Quality Labs and subsequently become listed in Microsoft Update. Overall, this would provide a much better experience for customers and developers choosing to implement this product.

Section 6: Developer & Customer Perspective

Introduction

This section assesses the experience a customer or developer would have, using the hardware keys. For the developer portion, testing will focus on securing a product with the keys while the customer perspective will analyze use of a product that is secured by these keys. The report shows the strengths and weaknesses of each device when evaluating keys for integration. This assessment examines the following details:

Customer Perspective

- Documentation required to use the product
    - For example, can the drivers for the key be integrated into the installer?
- Cleanliness of the installation/de-installation
    - Does the installer place files in a standard location?
    - Does the uninstaller remove key software?
- Appropriateness of error, alert and information messages
- Effort required by the user
    - Extra steps to be documented

Developer Perspective

- Security integration methods (wrapper, API)
- Integration documentation, such as tutorials online or included with developer's kit
- Ease of obtaining a developer's kit
- Support available for developers
    - Website
    - Support forums
    - Telephone (800 vs. Toll call)

Features

The USB devices can be judged on these criteria:

- Relative Advantage – what makes one device better than another?
- Complexity – How easy or difficult is it to use?
- Compatibility – Does it work with our existing procedures, policies, and equipment?
- Trialability – Can we test it before deployment? (This refers to development kits.)
- Observability – How does using one product versus another make us appear? (This correlates to the outward customer image.)

Each of these areas can be weighted equally, because they are all of equal importance when considering adoption of a new technology.

Relative Advantage for the Developer

Experiment 1.1: Comparison to Traditional Security Models

How does using hardware token authentication provide a more secure distribution of software?

Assertion

Hardware token authentication devices, like the Aladdin HASP HL and SafeNet Sentinel Hardware Key, provide a simple-to-manage protection scheme that is easy to distribute with commercial software. The devices do not require users to remember or store lengthy serial numbers, which can be easily distributed, nor do they require collection of personal information to validate a unique installation. Hardware keys, with their higher cost and perceived value, attach more value to the license and reduce the actual software media to a commodity.

HASP 100
Sentinel 100

Experimental Design

Identify the methods of securing a distributed application and weigh the benefits and disadvantages of each type.

Analysis

Security methods can be simply boiled down to these three categories:

1) What you know
2) What you have
3) Who you are

"What you know," describes items that can be documented with the packaging, such as numbers or codes. Typically, in software distribution, this is a serial number or key code. Often, keys will have checksum values to ensure authenticity when used to install the software.

"What you have" refers to tangible objects that must be referenced upon installation or each use. One such method is trivial lookup. This method was employed by the early 1990s video game X-Wing, where text from the user guide had to be entered on each usage. Another method from the late 1980s and early 1990s prompted the user to insert a program disk from the distribution into the drive at certain intervals during program usage. This method was used by software distributor Bullfrog for titles such as Theme Hospital.

"Who you are" refers to biometric information that identifies the user. While this could provide very secure software distribution, it also invites a very high level of overhead, not to mention expensive hardware for measuring fingerprints, retinas, etcetera.

Today's implementation of hardware token security falls into the "what you have" genre. With the cost to create exact digital duplicates of software media, efforts to improve "what you have" security have produced a non-replicable hardware device. A USB hardware token represents a higher cost, non-replicable device. It has the benefit of being tied to either a single user who carries the USB token, or a single machine with a token secured to one of its ports. These devices work in the same vein as previous parallel port security tokens, but take advantage of new PC specifications which often eliminate parallel ports in favor of more modern USB 1.1 and 2.0 access.

Complexity for the Developer

Experiment 2.1: Security by Shell Required Steps

What are the minimum steps required to lock an application using a shell/wrapper?

Assertion

The SafeNet Sentinel Hardware key provided a more streamlined process to create a simple shell-protected application. The sections in the Quick Shell application are numbered to easily guide the user through the required steps. The HASP HL-secured application displayed a cryptic "Feature Not Found" error when the limit of executions was reached; the Sentinel Hardware Key message was a much more user friendly "Execution Limit Exceeded." These messages can be modified by both key toolkits.

HASP 84
Sentinel 100

Experimental Design

The team installed the HASP toolkit and SafeNet License Designer from the media provided. From the standpoint of a developer with sufficient knowledge to navigate Microsoft Windows, the reviewers locked an application using the included tutorials. The researchers recorded the number of steps, number of decisions, and any additional options available along the way. In order to maintain compatibility between platforms, the Windows XP SP2 executable wordpad.exe was used.

Analysis

HASP HL approach

1) Launch HASP HL Vendor center from the Start Menu/Program/Aladdin/HASP HL/Vendor Center (note: Documentation is missing the Aladdin sub-folder.)
2) Insert the Master HASP key into the USB port on the computer.
3) Upon initialization, the HASP HL Envelope will detect the Master HASP HL key and require confirmation of the appropriate vendor code. This will only happen the first time the software is installed on the workstation.
4) Designate which programs to protect by dragging their icons into the right pane of the HASP HL Envelope GUI.
5) Set the program parameters. The user may customize error message, specify the frequency of key checks, etc.
6) Select the name of the application to protect from the left task pane.

7) Set the “Program number” to an unused value for the key in the Protection Details section.
8) Click Protect in the lower right corner of the window.
9) Launch the HASP HL Factory.
10) Right-click “Features” below the appropriate vendor ID.
11) Choose New Feature.
12) Enter a name, such as “Count Feature,” and set the program number to a value specified in step 7. Click Save.
13) Right-click “Orders” in the navigation pane.
14) Choose New Order.
15) Right click in the Features window and choose “Add Feature.”
16) Select the “Count Feature” created in the previous step. Click OK.
17) Select Counter from the “HASP License From” dropdown.
18) Select the radio button labeled “Set” and uncheck the “Unlimited” checkbox.
19) Enter the number 5 in the Activations control. Click OK.
20) Click the Save icon.
21) Click the Execute Order icon.

Sentinel Hardware Key approach

1) Click on Start\Programs\SafeNet Sentinel\Sentinel Keys 1.0\English\Toolkit.
2) Insert the SafeNet Sentinel Developer Key and User Key.
3) Select the radio button “Protect my application with the licensing limit I choose from below” in the licensing options section.
4) Check the checkbox “Execution Count.”
5) Enter “5” in the text field for “Execution Count.”
6) Click Prepare Key.
7) Select the source path for the application chosen to lock.
8) Select the destination path for the locked executable.
9) Select the Sentinel Key to bind the application, using the controls in the Key Status frame.
10) Click Make Shell.
11) Click Make Key.

Experiment 2.2: Security by API Required Steps

What steps are required to lock an application using the API using Visual Basic .NET?

Assertion

The Aladdin API toolkit was simple and efficient. It provided a sandbox environment where known-good code was tested on a key to ensure functionality, and the copy and paste methods made for rapid code delivery. It was also beneficial that the samples could be easily downloaded from Aladdin's website, in addition to the toolkit for developers who are not in possession of the SDK.

The Sentinel SafeNet Hardware Keys SDK contained a toolkit to generate code, but did not include any code samples to show implementation. It also asked for a Developer ID in hexadecimal, although it was not obvious where this could be obtained. Trial and error eventually determined it. The API toolkit also offered a help section, but it was not searchable and actually worked more like a table of contents.

HASP 100
Sentinel 34

Experimental Design

The research team installed the HASP toolkit and SafeNet License Designer from the media provided. From the standpoint of a developer with sufficient knowledge to navigate Microsoft Windows, the team locked a vb.net application (using API calls) by following the included tutorials. The researchers recorded the number of steps required to initialize the key, store a value and retrieve the same value.

Analysis

HASP

1) Launch the HASP Vendor Center.
2) Insert the Master key.
3) Set a language preference in Toolbox (C#, VB) using File>Settings.
4) Select the hasp.login() function.
5) Choose the appropriate vendor code.
6) Copy and paste the data into your Visual Studio .NET project.
7) Add the following line to the top of your .vb file:

```
Imports Aladdin.HASP
```

8) Right-click on the project name in the Solution Explorer.
9) Choose "Add Reference."
10) Add this file from the HASP-HL CD:
    D:\Windows\Installed\API\Runtime\dotnet\hasp_net_windows.dll
11) Select hasp.write in the HASP-HL toolkit.
12) Copy and paste the resulting code.
13) Select hasp.read in the HASP-HL toolkit.
14) Copy and paste the resulting code.

Supporting Data

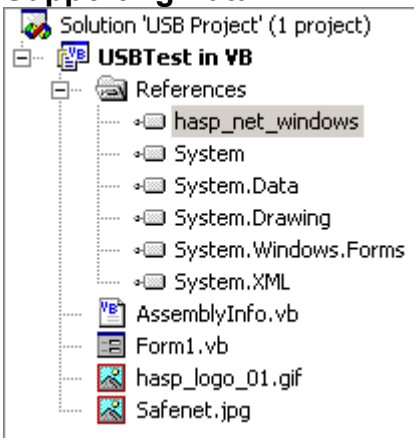


Figure 2.2.1 – HASP DLL in Visual Studio .NET 2003

Experiment 2.3: Multiple Application Security

Can one key be used for multiple applications?

Assertion

Both keys were easily able to support multiple applications with different options on each key. For the same reasons outlined in experiment 2.1, the HASP application required more steps.

HASP 84
Sentinel 100

Experimental Design

The researchers created a shell-secured application (wordpad.exe) using the tools provided by the vendor. The team installed this package on a system that did not have any tools installed on it. Using the same key, the team protected a second application (notepad.exe) with the supplied tool kits. The researchers ensured that both the first and second applications could still be executed when the hardware key was present. The first application should have a 5-count limit, and the second application should be designated as unlimited execution.

Analysis

According to the packaged documentation, the Aladdin HASP HL Max key can support up to 112 unique licenses on a single hardware token. These licenses may refer to an individual application or a unique feature within an application.

The Sentinel License Designer permits a developer or distributor to create a license type that has multiple features, or applications. Each shelled application or API can be added to a custom license using the Designer with his or her own unique options, such as iteration count or timed-expiration. The limit on applications appears to be memory-bound, not license-count bound as was found with the Aladdin HASP HL.

Compatibility for Developers

Experiment 3.1: Driver Installation/Removal

Does the installer provide a clean installation and removal of drivers and support software?

Assertion

Both software toolkits provide simple, straightforward installers and uninstallers. The Sentinel installer provided a cleaner uninstall, leaving no trace of its existence on the system. It would have been better if the uninstaller had been available on the start menu.

Experimental Design

Many files and registry entries are made by the software installer when new software is added to a system. Using Total Uninstaller 3.1, each software package was installed and immediately removed. Log files from Total Uninstaller were analyzed to see which files or registry entries were added, removed, or updated.

Analysis

HASP has an uninstaller available in its own Start Menu group, as well as within the add/remove programs control panel. The following files and folders were orphaned by this program:
c:\program files\aladdin\hasp HL\vendortools\vendorcenter\db\
c:\program files\aladdin\hasp HL\vendortools\vendorcenter\db\monster.mdb
%windows%\system32\strtstop.dll
%windows%\system32\hasp_windows.dll
%windows%\system32\hasp_msi.dll

SafeNet Sentinel did not have an uninstall feature on the start menu, but one was found within the add/remove programs control panel. There were no questions or prompts. No files and folders were orphaned by this program.

Scoring

HASP 90
Sentinel 100

Trialability

Experiment 4.1: SDK Procurement

How are SDKs obtained on a trial basis? What are some of the issues or concerns in obtaining the kits, including cost?

Assertion

Both vendors offer SDKs through an easy-to-use web interface that is directly linked from the product information page.

Experimental Design
The researchers procured a development kit from both vendors. The team recorded the turn-around time for both kits, detailing the procedure involved in obtaining the kits, including the number of phone calls. Metrics included price, delivery days, steps to order, contacts required.

Analysis
Aladdin offers a form on their website to request an SDK. This form is followed up by a call from a sales specialist who evaluates the individual request and then sends the SDK free of charge.

SafeNet also offers an SDK kit through a web form interface. This report cannot confirm the length of time to deliver or the presence of a follow-up call because insufficient time was allotted for the experiment.

Scoring
HASP 100
Sentinel 100

Experiment 4.2: Support Options
What types of support are available to the developer during a trial phase? Is additional support available after a purchase has been completed?

Assertion
Aladdin provides better support to the developer with more attentive sales support, technical support, and quality documentation. SafeNet provides better hours of support, which is beneficial for companies located outside the US.

Experimental Design
The researchers visited website for both vendors. The team documented a list of the features available including:

- Telephone
    - Hours
    - Type of number
    - Applicable charges
- Support forums
    - Anonymous vs. registration
- FAQ's/Support Library
    - Quality of documentation
- E-mail support
    - Turn around time
    - Escalation structure

Analysis

	Aladdin	SafeNet
Telephone	Toll Free Number available 1-866-202-3494 No phone prompts, answered directly by a person. East coast, available 9a-6p No charge for phone support or e-mail support	Toll Free Number available 1-800-545-6608 (prompt 1) 24x7 phone support No charge for phone or e-mail support 443-327-1242 Andy Grimada
Forums	None	None
FAQ's/Library	Only 4 for HASP HL, more available for HASP4	Search for "Sentinel Hardware Key" and "Sentinel UltraPro" found 20 items available.
Email Support (Form)	E-mail form generated a sales call. Typical turn around is 24 hours.	E-mail form does not work. Tests attempted on Windows XP SP2 with Firefox 1.5.0.3 and Internet Explorer 6. In addition, the form does not validate any e-mail addresses from any country TLDs (such as co.uk or .us)
Email Support (Direct)	Unable to locate	support@safenet-inc.com

Phone: This experiment began by contacting both technical support centers and evaluating the support options available for developers. The phone representative at Aladdin gave a company greeting and his name. The team stated that they were evaluating the HASP HL hardware tokens and identified themselves with RIT. The representative used active listening techniques to confirm the questions asked, and then gave appropriate answers. He was unable to provide any pricing information on a starter kit or user token pricing.

The next call to SafeNet began with an automated prompting system and a greeting, which advertised www.safenet.biz. The researcher pressed "1" for technical support, which was immediately answered. The technician identified the company but not his own name. The researchers identified with the same information as the Aladdin phone call, this time substituting Sentinel Hardware Keys. The technician misinterpreted the information and began a script to get callback information. He stated that an engineer more familiar with the Sentinel Hardware Keys would return the phone call shortly. The researcher refuted that he merely wanted to know what technical support options existed, and that he was having no technical issues with the keys. The technician was able to tell him that SafeNet provides 24x7 support using call centers around the world, and a sales representative would be more suited to answer any other questions. When asked to transfer the call, he transferred it to general phone line in the North Carolina sales office.

The person who answered that line suggested contacting the local sales representative and provided their local toll number. Upon calling that number, it rolled directly to voice mail, and the team did not choose to leave a message. Further calls to the same number also resulted in voice mail, and on the fourth attempt, a message was left.

E-mail: The e-mail form was filled out on the Aladdin website requesting technical assistance. Since the user name on the form did not match an existing registered company in Aladdin's database, a sales rep contacted us. He inquired where the keys were obtained, and then gathered that information. Although not registered customers, the sales rep was willing to answer questions.

The e-mail form on the SafeNet site resulted in an error, shown as figure 4.2.1.

FAQ's/Library: Neither website had user forums that the team was able to navigate. It is possible that additional documentation would have been available with the appropriate customer credentials.

On both websites, the documentation about the most recent hardware tokens was very sparse – only four documents from Aladdin and two drivers from SafeNet. The scope was expanded to previous generation hardware for both manufacturers to analyze the documentation quality, with the hypothesis that any documentation produced by a company would be representative of future documentation quality. After importing five documents from both SafeNet and Aladdin into Microsoft Word, they were each spell checked and analyzed for grammar and readability. No spelling errors were detected in the Aladdin documents. Multiple spelling errors, including product names, were detected in the SafeNet documents. In many cases, the sentences were fragmented or entirely unreadable. Some examples are shown below in figures 4.2.2 and 4.2.1

Supporting Data

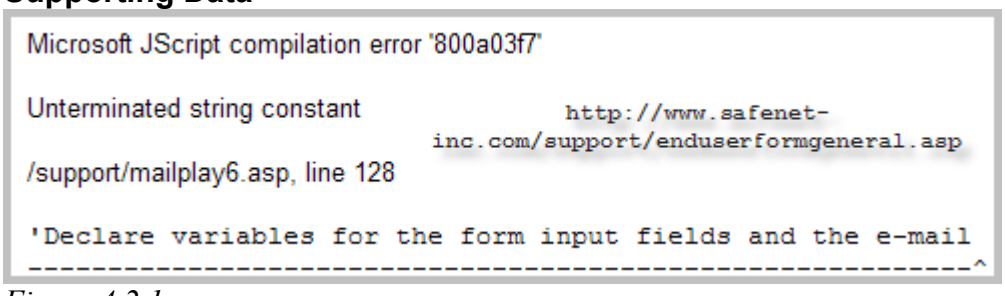
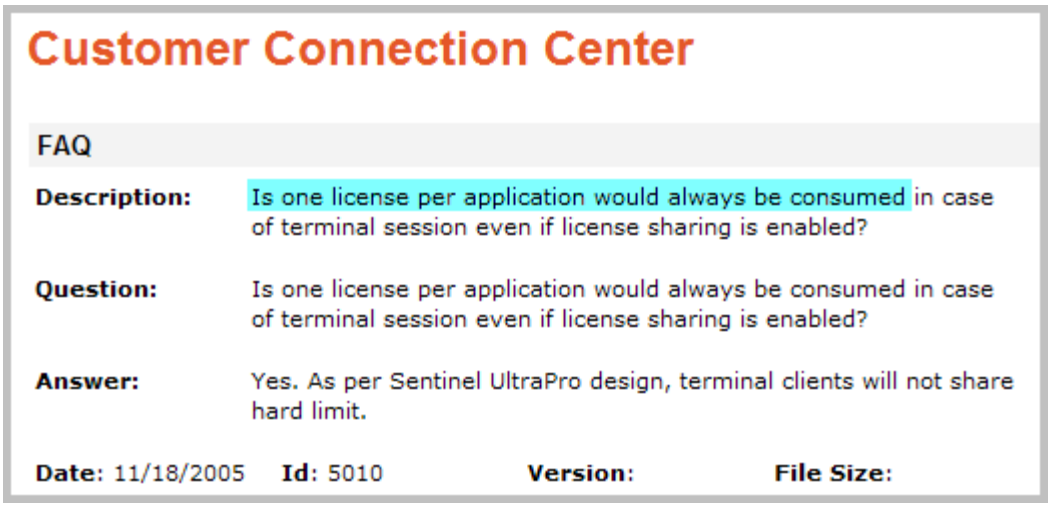


Figure 4.2.1



Customer Connection Center

FAQ

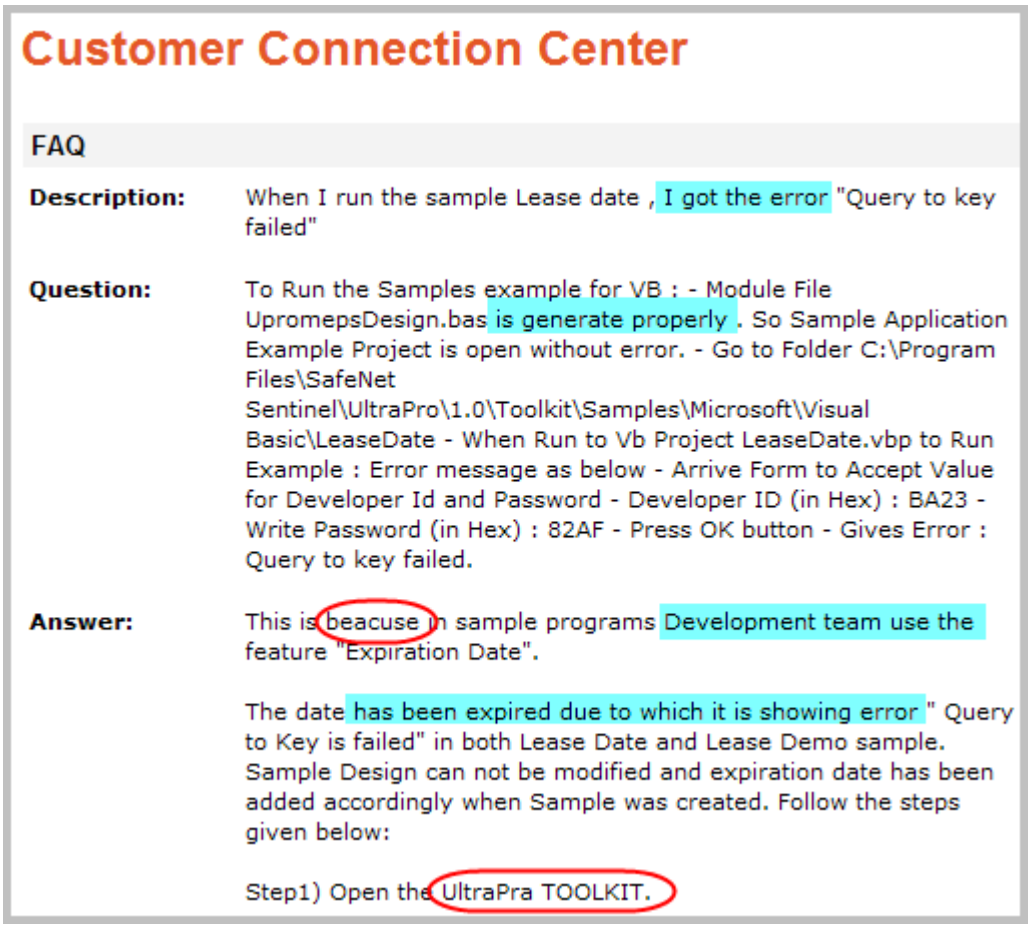
Description: Is one license per application would always be consumed in case of terminal session even if license sharing is enabled?

Question: Is one license per application would always be consumed in case of terminal session even if license sharing is enabled?

Answer: Yes. As per Sentinel UltraPro design, terminal clients will not share hard limit.

Date: 11/18/2005 **Id:** 5010 **Version:** **File Size:**

Figure 4.2.2



Customer Connection Center

FAQ

Description: When I run the sample Lease date , I got the error "Query to key failed"

Question: To Run the Samples example for VB : - Module File UpromepsDesign.bas is generate properly . So Sample Application Example Project is open without error. - Go to Folder C:\Program Files\SafeNet Sentinel\UltraPro\1.0\Toolkit\Samples\Microsoft\Visual Basic\LeaseDate - When Run to Vb Project LeaseDate.vbp to Run Example : Error message as below - Arrive Form to Accept Value for Developer Id and Password - Developer ID (in Hex) : BA23 - Write Password (in Hex) : 82AF - Press OK button - Gives Error : Query to key failed.

Answer: This is beacuse in sample programs Development team use the feature "Expiration Date".

The date has been expired due to which it is showing error " Query to Key is failed" in both Lease Date and Lease Demo sample. Sample Design can not be modified and expiration date has been added accordingly when Sample was created. Follow the steps given below:

Step1) Open the UltraPra TOOLKIT.

Figure 4.2.3

Scoring

HASP 100

Sentinel 40

Observability for the Developer

Experiment 5.1: Replacement Tokens

What occurs with an application when a hardware token fails? Can "backup" or duplicate keys be made for a customer for a specific application?

Assertion

Creating a duplicate key within the HASP-HL Max starter kit was easy, but the values and counters were reset. Both vendors had replacement methods available for failed keys, but SafeNet did not return the team's call for p[illegible]g or timing.

Experimental Design

Part 1: The research team contacted the support line at Aladdin and SafeNet to inquire about the warranty options for failed keys.
Part 2: The team secured an application (wordpad.exe) using a single hardware token. The researchers used the application, and marked that key as "original." They removed the key and set it aside as "broken." The researchers then determined the steps to create a new key that could still operate that same software.

Analysis

Using the HASP HL Factory, the order that has the protection features for a specific set of application(s) can be written to another HASP user key, and the protected applications will run from either of the two keys. If any of the applications has a time limiting feature on it, the limits are reset when writing the order out to a different user key. Any execution counts will return to their initial values, and any "time bombs" will be reset to the time when the new key was written.

Using the supplied documentation, the team was unable to determine how to initialize a duplicate key. The supplied test kits contained a single user key, developer key and distributor key. User keys from a different kit appeared not to be compatible with the developer key from the first kit. Since an incomplete comparison could not be made, this fact is not represented in the scoring of the SafeNet product.

Per the team's conversation with the sales representative from HASP, replacement keys are available under warranty within a short time frame, often in 24-48 hours. Expedited keys can be made available for mission critical applications. For keys with user damage, keys can be purchased from the software vendor or distributor.

A message was left on the voice mail for the SafeNet sales representative responsible for our geographical region. He did return our call and leave another voice message, but the call was outside of the period of this report.

Supporting Data

Experiment was performed over the telephone.

Scoring
HASP 90
Sentinel 80

Relative Advantage for the End User

Experiment 1.1: Comparison to Traditional Security Models
What benefits does the end user of high-value/low-volume software using hardware token authentication experience over other security methods? What are the disadvantages?

Assertion
When using a hardware token to authenticate, the user is locked out of the application unless the token is present. However, one can install and use the application on as many computers as desired.

With serial number authentication, the end-user license agreement (EULA) usually allows the software to be installed on one computer. The serial number is distributed in some other form, adding another part of the total package (software + serial number) to run the application.
With product activation, the software must "phone home" in some way to the developer to allow the software to run.

Analysis
There are many benefits for using a hardware-based authentication token. First, the license for the product is stored on the key, allowing installation of the software on as many computers as desired at one time, legally. To use said program, one needs to have the authentication token present, which prevents multiple copies of the program from running at once.

The advantages that the authentication token has over other security methods such as serial numbers, pass phrase lookups, and product activation is that only the authentication token is required for the software to work. Nothing else needs to be remembered at the time of installation or use. This is a clear advantage over serial numbers, where one must remember a string of alphanumeric characters ranging as many as 30 characters. Serial numbers are generally stored on the CD case, which is another physical item besides the CD that can get lost, stolen, or broken. There is no user manual needed. Product activation requires either an active internet connection or a call to the company where information is collected about the user who is registering the product. This information has a chance to be intercepted by malicious third parties when it is sent across these mediums.

The disadvantage of using a hardware-based authentication token is that the key needs to be present for the software to run. If the key is lost or broken, the software cannot be used again unless another binary is distributed locked with another key. If a serial

number is lost, one can be re-requested (neither HASP nor Sentinel would provide pricing details on replacement keys) from the developer of the software. Serial numbers are also vulnerable to internet groups releasing a key generator for the software, which allows anyone to obtain a valid serial number for that program. In the case of pass phrase lookups, a table or text file can be made with lists of all the questions and answers, or a scan of the reference material can be made. Product activation can usually be defeated by using a key generator for programs that have telephone authentication that will generate a proper response code.

Scoring
HASP 100
Sentinel 100

Compatibility for the End User

Experiment 2.1: Runtime Limitations
How many instances of a program that is secured can be run at once? Do they all have to be run from the same machine and the same user? Can multiple users use secured applications from the same machine (on a terminal server)? Can the keys be queried across the network?

Assertion
Unless the developer specifically denies it, multiple copies of the same locked software can be run multiple times from the computer as the same user or different users. The Sentinel keys can be queried across the network if the keys are set up on a broadcasting server. Applications cannot be used via Terminal Services.

Experimental Design
- The researchers locked a program using both Sentinel and HASP, and then tried to launch multiple concurrent copies of the same locked application.
- The team used the "Run As" command to launch multiple local copies using different users.
- Under Windows 2003 Server, the team set up Terminal Services to see if more than one *remote* user could run an application while the USB token was in the server machine.
- The team set up two machines to see if machine A could query the key on machine B to run the program on machine A.

Analysis
Programs locked by Sentinel were able to run multiple copies of executable, as along as the developer did not specifically disable this function. The processes did not have to be run all from the same user, using the "Run as…" command in Windows 2000 and above; two users could run the same application. However, this was not very useful, as two users could not use the same system at once. Figure 2.1.1 shows two copies of

WindowsApplication1_SHELLED being run by the Administrator user, and Figure 2.1.2 shows two copies of the same executable being run by Administrator and John. Programs locked by HASP were also able to run multiple copies of the executables, from the same user or two different ones by using the "Run as…" command. Figure 2.1.3 and Figure 2.1.4 illustrate this.

Both Sentinel and HASP were able to query keys across the network. However, the developer kits that the team received for both Sentinel and HASP lack this feature, so the test could not be preformed.

Sentinel applications could not be run on a Terminal Server machine at all. After installing Terminal Services, trying to run an application that was successfully run on the same machine before TS was installed, an error that the keys could not be found was displayed (as shown in Figure 2.1.5). HASP protected programs could be run from Terminal Services, only if the keys supported it and the functionality was programmed in. With the keys that were available to be tested, when launching a HASP program while logged in via Terminal Services, HASP saw that it was running in a virtual environment and displayed an error, shown in Figure 2.1.6.

Supporting Data

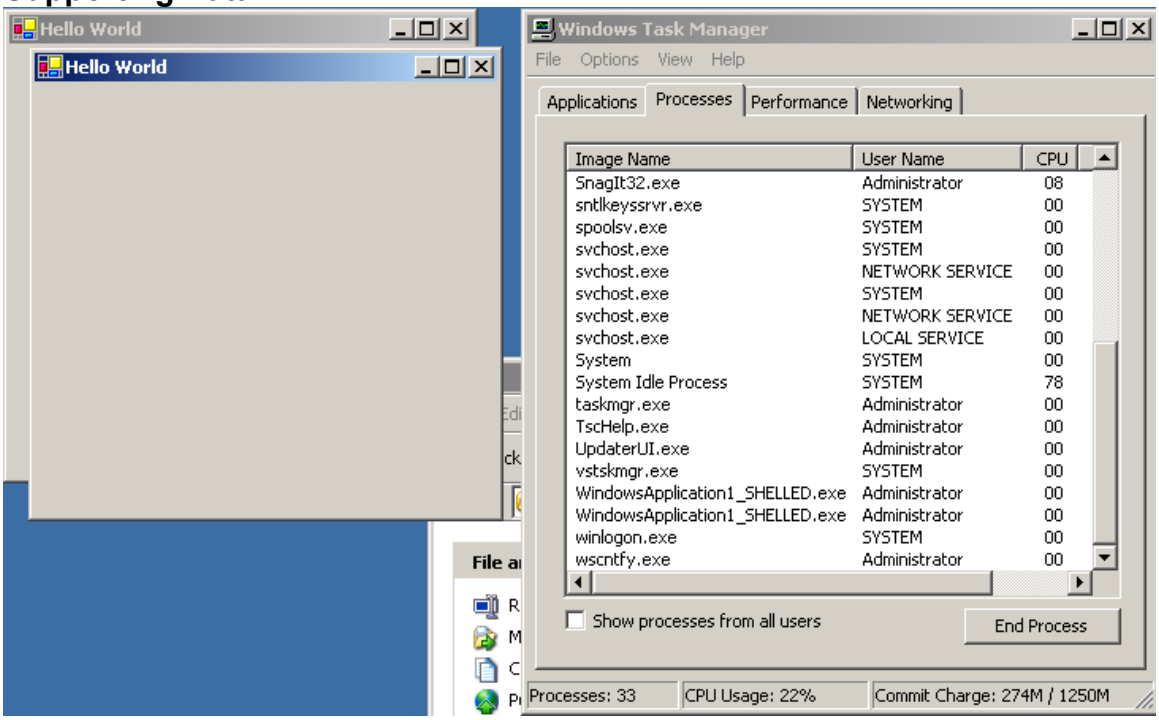


Figure 2.1.1 – Two copies of WindowsApplication1_SHELLED, wrapped in a Sentinel Envelope, being run locally by the Administrator user.

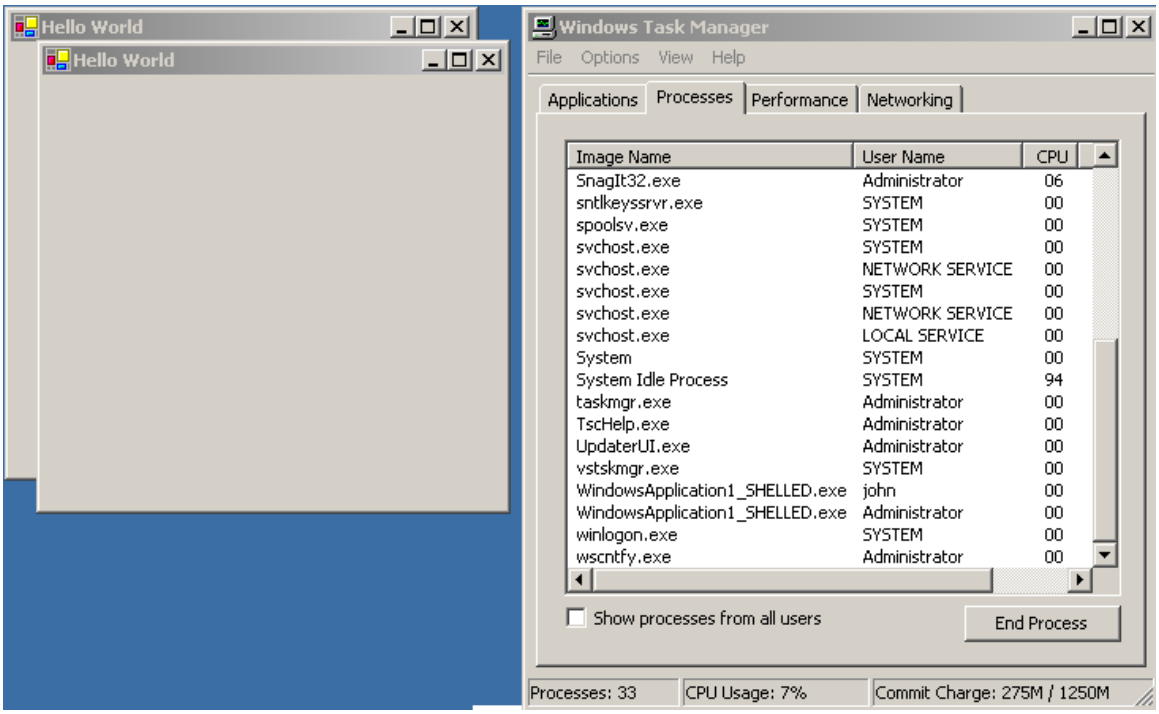


Figure 2.1.2 – Two copies of WindowsApplication1_SHELLED, wrapped in a Sentinel Envelope, being run locally by the Administrator and John users.

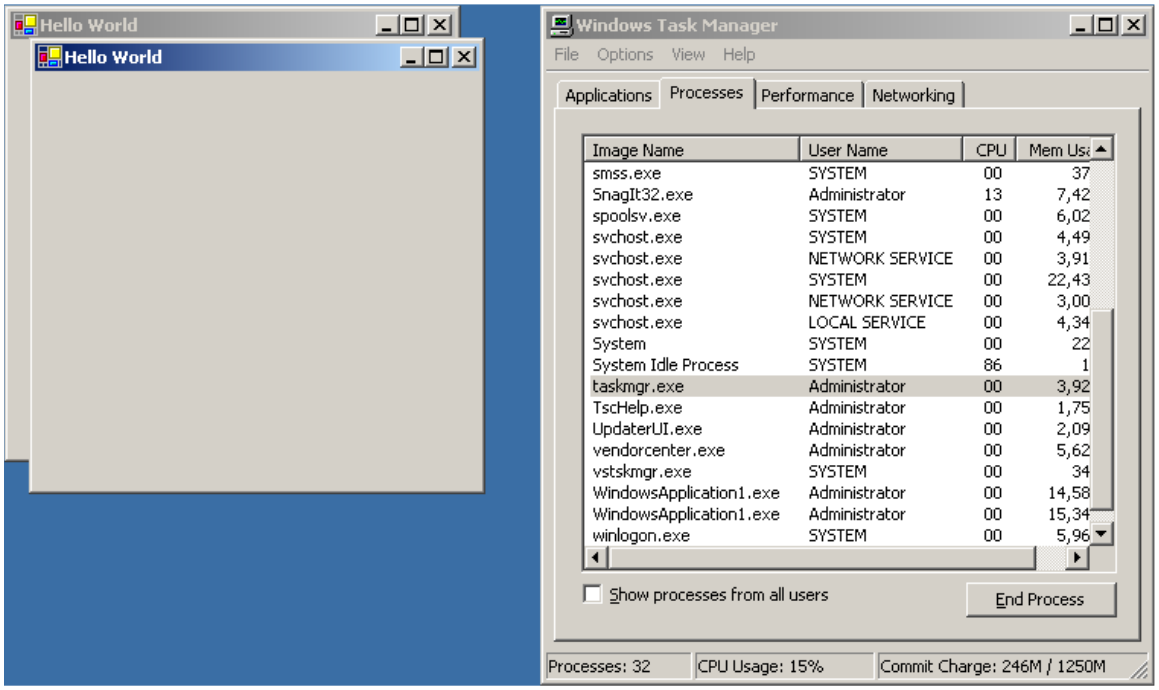

Hello World
Hello World
Windows Task Manager
File Options View Help
Applications Processes Performance Networking
Image Name User Name CPU Mem Usa
smss.exe SYSTEM 00 37
SnagIt32.exe Administrator 13 7,42
spoolsv.exe SYSTEM 00 6,02
svchost.exe SYSTEM 00 4,49
svchost.exe NETWORK SERVICE 00 3,91
svchost.exe SYSTEM 00 22,43
svchost.exe NETWORK SERVICE 00 3,00
svchost.exe LOCAL SERVICE 00 4,34
System SYSTEM 00 22
System Idle Process SYSTEM 86 1
taskmgr.exe Administrator 00 3,92
TscHelp.exe Administrator 00 1,75
UpdaterUI.exe Administrator 00 2,09
vendorcenter.exe Administrator 00 5,62
vstskmgr.exe SYSTEM 00 34
WindowsApplication1.exe Administrator 00 14,58
WindowsApplication1.exe Administrator 00 15,34
winlogon.exe SYSTEM 00 5,96
Show processes from all users
End Process
Processes: 32
CPU Usage: 15%
Commit Charge: 246M / 1250M

Figure 2.1.3 – Two copies of WindowsApplication1, protected by HASP, being run locally by the Administrator user.

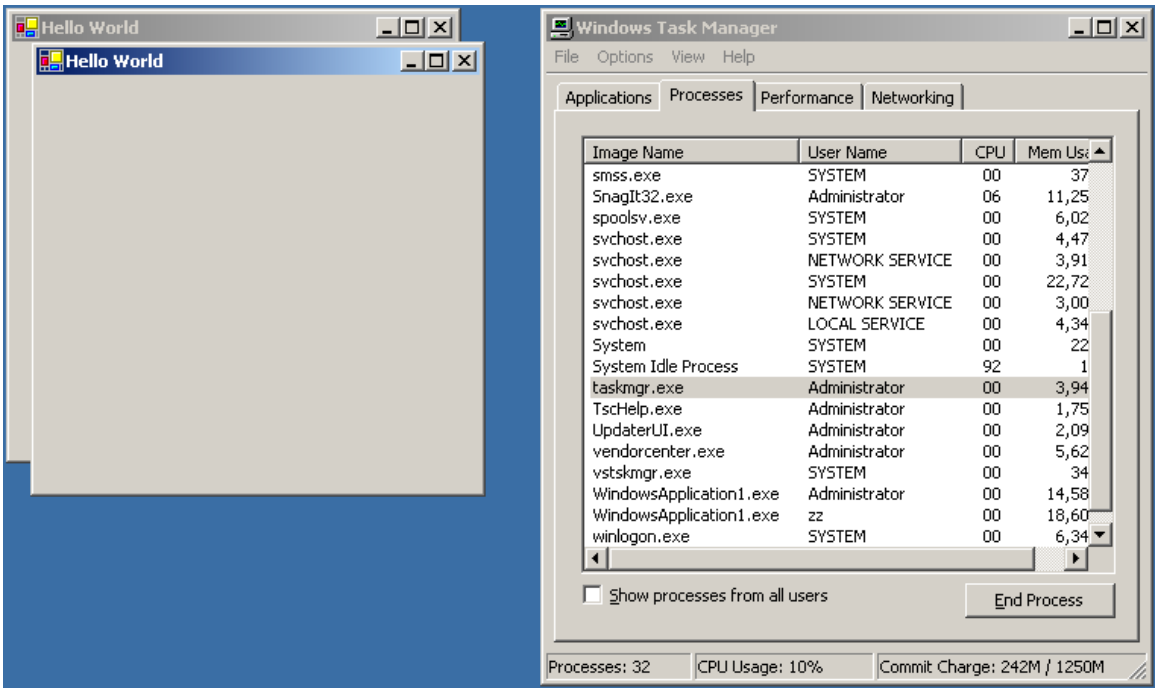

Hello World
Hello World
Windows Task Manager
File Options View Help
Applications Processes Performance Networking
Image Name User Name CPU Mem Us
smss.exe SYSTEM 00 37
SnagIt32.exe Administrator 06 11,25
spoolsv.exe SYSTEM 00 6,02
svchost.exe SYSTEM 00 4,47
svchost.exe NETWORK SERVICE 00 3,91
svchost.exe SYSTEM 00 22,72
svchost.exe NETWORK SERVICE 00 3,00
svchost.exe LOCAL SERVICE 00 4,34
System SYSTEM 00 22
System Idle Process SYSTEM 92 1
taskmgr.exe Administrator 00 3,94
TscHelp.exe Administrator 00 1,75
UpdaterUI.exe Administrator 00 2,09
vendorcenter.exe Administrator 00 5,62
vstskmgr.exe SYSTEM 00 34
WindowsApplication1.exe Administrator 00 14,58
WindowsApplication1.exe zz 00 18,60
winlogon.exe SYSTEM 00 6,34
Show processes from all users
End Process
Processes: 32
CPU Usage: 10%
Commit Charge: 242M / 1250M

Figure 2.1.4 – Two copies of WindowsApplication1, protected by HASP, being run locally by the Administrator and zz users.

Figure 2.1.5 – The error displayed when trying to run a Sentinel protected application on a Windows 2003 machine with Terminal Services installed.

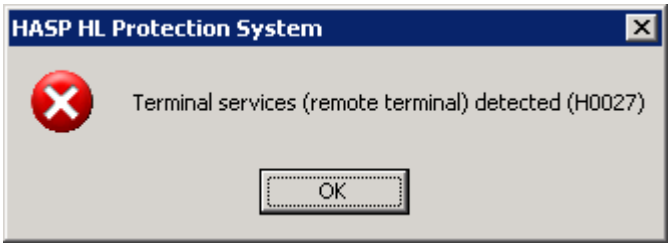


Figure 2.1.6 – The error displayed when running a HASP protected program in Terminal Services.

Scoring
HASP 100
Sentinel 40

Complexity for the End User

Experiment 3.1: User Interaction

What are the steps required for a user of an application, which is secured with a hardware token authentication system, to execute said application normally?

Assertion

In the case of a wrapped or shelled application, the key must be inserted into the machine before the program is launched, as there are calls made to the key when the binary is initially run to verify that the key is present.

In the case of API calls, whenever there is a call in a section of code that requires something from the key, the key must be present. There are calls available to query whether the key is present at specified intervals.

Experimental Design

The research team locked an application using both Sentinel and HASP. The team wrapped all the files needed to launch the protected application together in an installer. It deployed the installer package to a separate computer and document each step required to launch the application successfully.

Analysis

For both Sentinel and HASP, after the installation packages are installed containing the drivers needed for each to run the keys and the protected application deployed, the user must insert his key into a USB slot. Once the machine recognizes and initializes the key, the user must then launch the application. Before the program is launched, the keys will be queried. If they are the correct keys, the program will launch.

Scoring

HASP 100
Sentinel 100

Trialability for the End User

Experiment 4.1: Trial Scope of Features

What are the restrictions that can be applied to an application used for trial purposes? What methods are there to extend this trial period?

Assertion

The program can be locked with a run count limit, after a certain amount of days, or until a specific time.

Analysis

Both Sentinel and HASP protected programs can be distributed with time-limited installations. Both products support the same three licensing models: limited number of program executions, unlimited use until a certain date, and a time limit from when the

applications are protected. When the time limit is up, the programs will refuse to run. The Sentinel application will display a dialog that says "Program usage count exhausted" or "Program Terminal Date/Time reached." The HASP application is a bit more cryptic, simply stating, "Feature not found." These dialog boxes can be seen in Figures 4.1.1 through 4.1.3.

To extend the trial period for a Sentinel protected application, the user must first open the Secure Update Utility and attach the USB key. Once the key is connected to the system, the user must click on Generate Request Code, which will poll the key for the information stored on it. The user is then present with a dialog box prompting him to save the Request Code File to a location of his choice. Once this is done, the user must contact the developer and request that his application license be extended, and send the Request Code File to the developer.

When the developer receives this request along with the Request Code File, the developer then opens a copy of the Update Manager, and selects the License/Feature radio button and clicks on Add. A new dialog entitled Add Action or Features is presented, and the developer will then click on Add Command. From the Commands dropdown box, the developer then selects the feature he wishes to add. A value is then specified and the developer clicks OK. At the Update Manager screen, on the Key Activator tab, the developer will then load the Request Code File and select the features to be added. Once all the features are loaded, the developer will click on the Generate update code button. Upon successful update, a new dialog will appear and the Update Code file must then be saved.

The developer then sends the Update Code file back to the user, and the user inserts his USB key and loads the Secure Update Utility again, opening the Update Code file. Once the file is loaded, the user clicks on Activate Application. Upon successful update, a dialog will appear.

To extend the license for a HASP protected application, one must first open the Vendor Center and select Envelope, and then add the program desired to modify the license for to the project. Under Protection Details, select an unused program number to store the protection feature, and then click on Protect to make the secured executable.

Next, open the Factory from Vendor Center. Under the developer code for the specific set of keys, click on Features, and then right click in the right pane and select New Feature. Specify a name for the feature chosen to add, and click on the Save icon to save it. Under Packages for the developer code, right click in the right pane and select New Package. Give this new package a name, and then right click in the lower features area and select Add Feature. Choose the form of licensing desired (in this case, a counter) and deselect "Unlimited" activations. Set the number of activations to any number, and click OK. Click on Orders, and then right click in the right pane and choose to add a New Order. Name the order, and then right click in the lower pane and choose to add the package just created. Once this is done, save the order, right click on it, and choose to Execute Order. Insert a user key, and this will write the licensing data to the user key.

Supporting Data

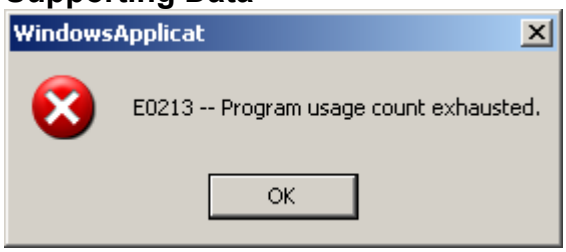


Figure 4.1.1 – The dialog shown when the user has exceeded their program execution limit in a Sentinel protected program.

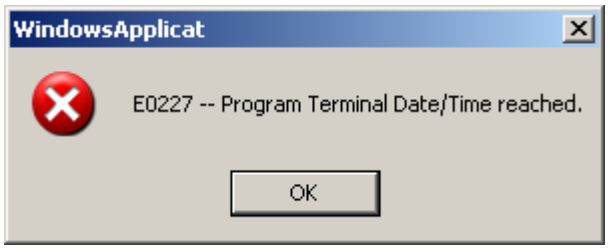


Figure 4.1.2 – The dialog shown when the application time limit has expired in a Sentinel protected program.

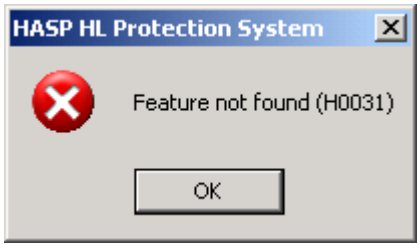


Figure 4.1.3 – The dialog shown when the user has exceeded their program execution limit or the time limit expires in a HASP protected program.

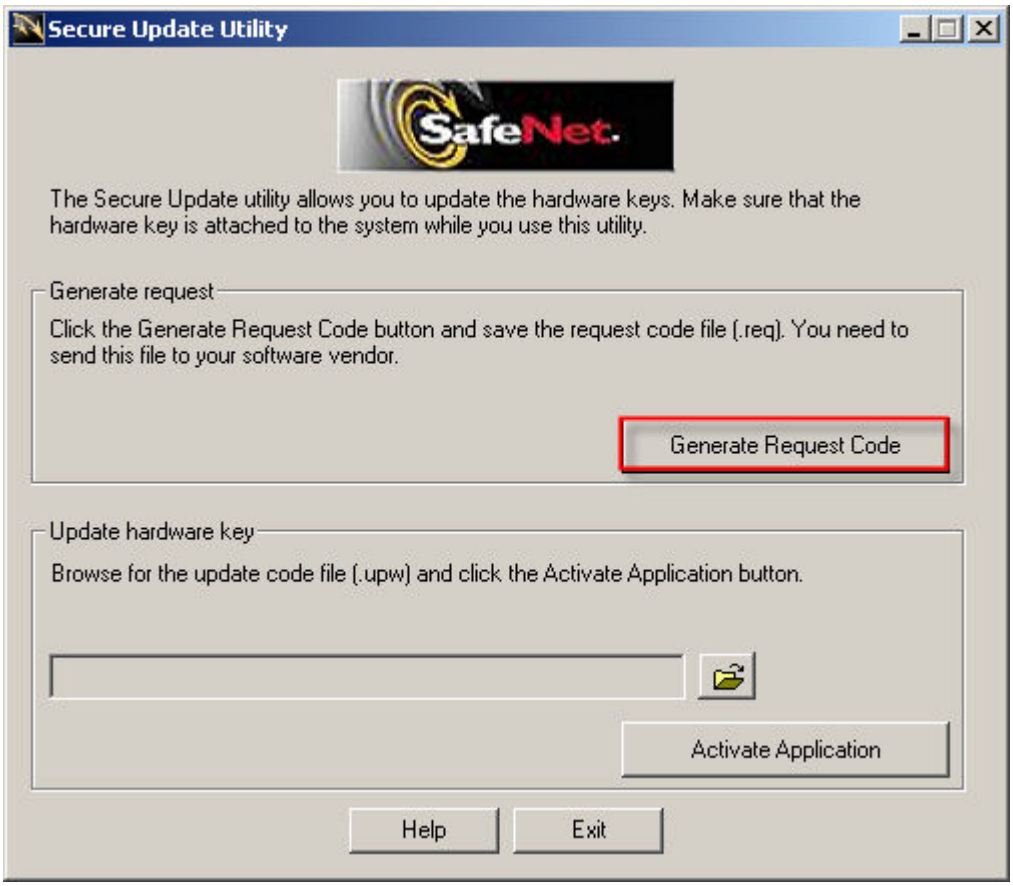

Secure Update Utility
SafeNet.
The Secure Update utility allows you to update the hardware keys. Make sure that the hardware key is attached to the system while you use this utility.
Generate request
Click the Generate Request Code button and save the request code file (.req). You need to send this file to your software vendor.
Generate Request Code
Update hardware key
Browse for the update code file (.upw) and click the Activate Application button.
Activate Application
Help
Exit

Figure 4.1.4 – The first step in updating the time limit license in a program. The user inserts their key and generates a request code.

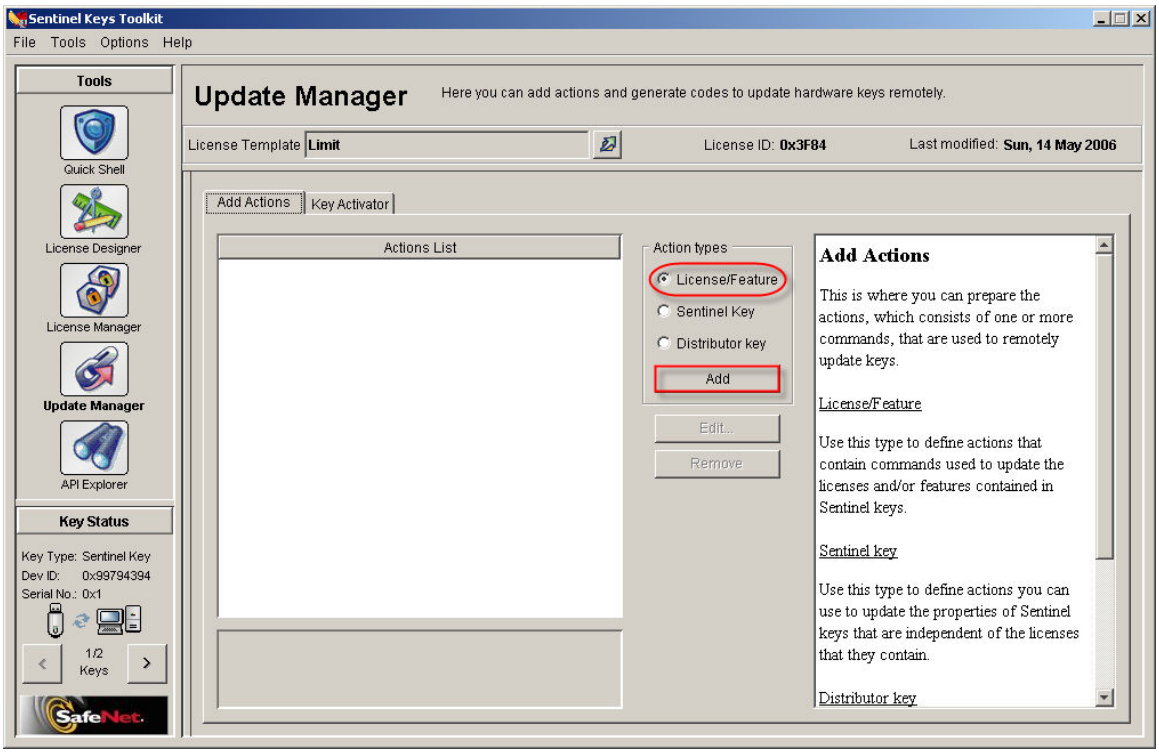

Sentinel Keys Toolkit
File Tools Options Help
Tools
Quick Shell
License Designer
License Manager
Update Manager
API Explorer
Key Status
Key Type: Sentinel Key
Dev ID: 0x99794394
Serial No.: 0x1
1/2 Keys
SafeNet.
Update Manager
Here you can add actions and generate codes to update hardware keys remotely.
License Template Limit
License ID: 0x3F84
Last modified: Sun, 14 May 2006
Add Actions
Key Activator
Actions List
Action types
License/Feature
Sentinel Key
Distributor key
Add
Edit...
Remove
Add Actions
This is where you can prepare the actions, which consists of one or more commands, that are used to remotely update keys.
License/Feature
Use this type to define actions that contain commands used to update the licenses and/or features contained in Sentinel keys.
Sentinel key
Use this type to define actions you can use to update the properties of Sentinel keys that are independent of the licenses that they contain.
Distributor key

Figure 4.1.5 – The Update Manager, where the developer is able to add in license adjustments.

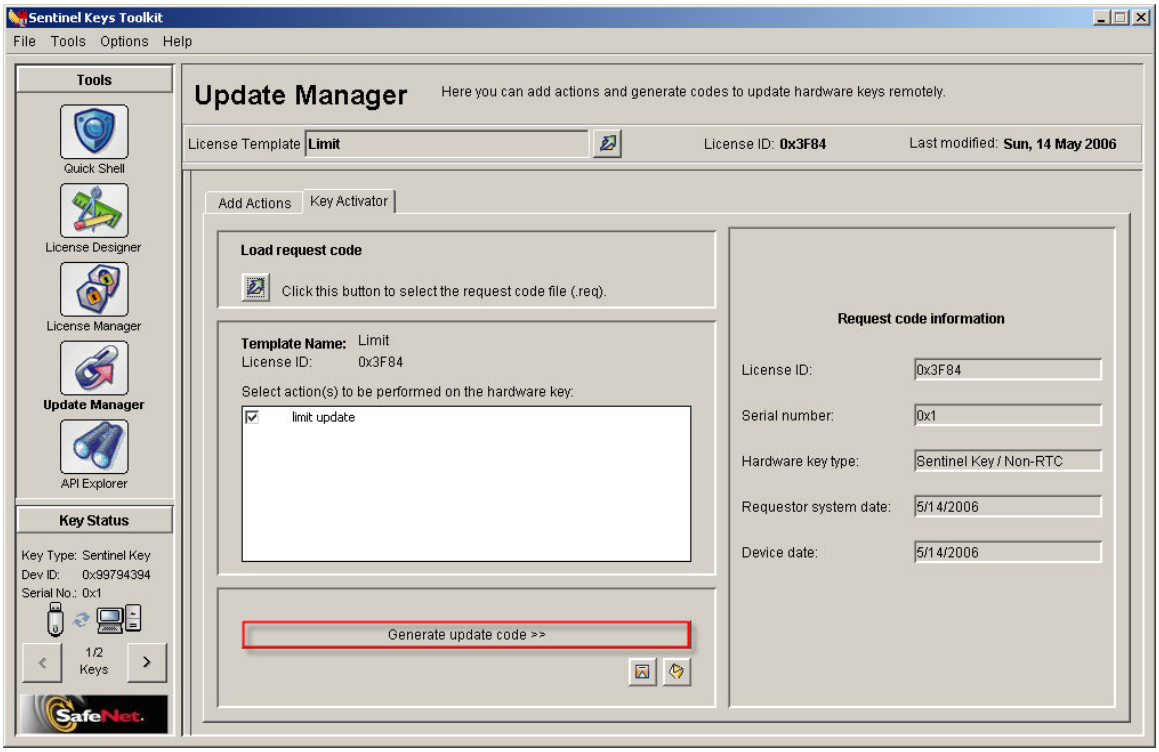


Figure 4.1.6 – The developer loads the user's request code, adds in the feature that adjusts the execution limit, and then generates an update code.

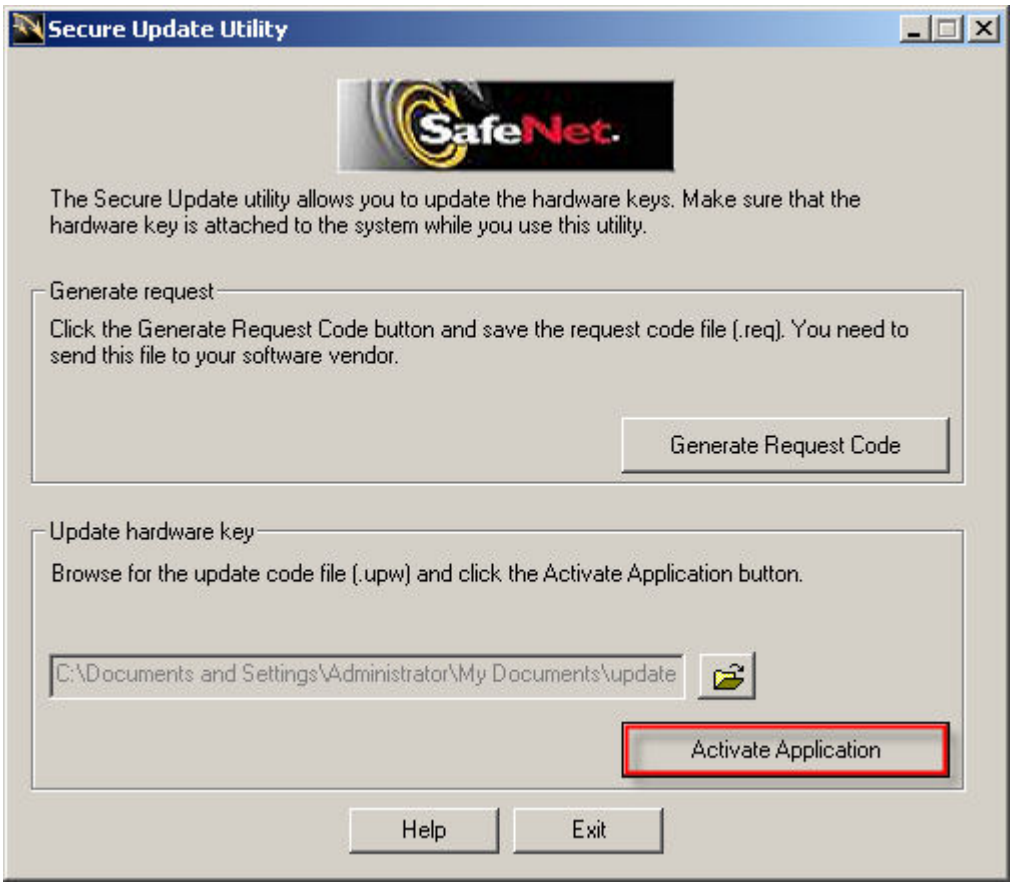

Secure Update Utility
SafeNet.
The Secure Update utility allows you to update the hardware keys. Make sure that the hardware key is attached to the system while you use this utility.
Generate request
Click the Generate Request Code button and save the request code file (.req). You need to send this file to your software vendor.
Generate Request Code
Update hardware key
Browse for the update code file (.upw) and click the Activate Application button.
C:\Documents and Settings\Administrator\My Documents\update
Activate Application
Help
Exit

Figure 4.1.7 – When the user receives the update code file from the developer, the user then loads the Secure Update Utility, loads the update code file, and attempts to activate the application.

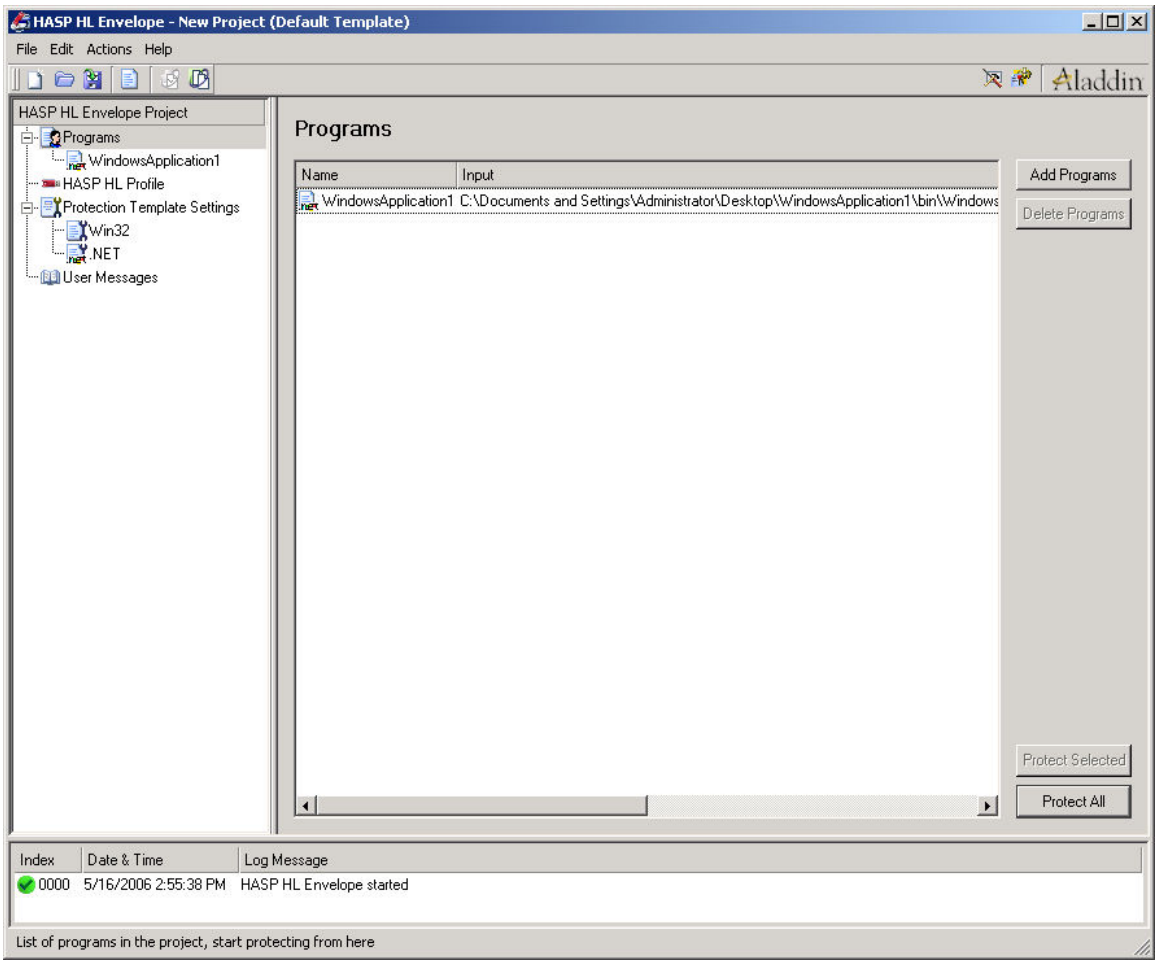


Figure 4.1.8 – Add the program to which you wish to edit the licensing scheme.

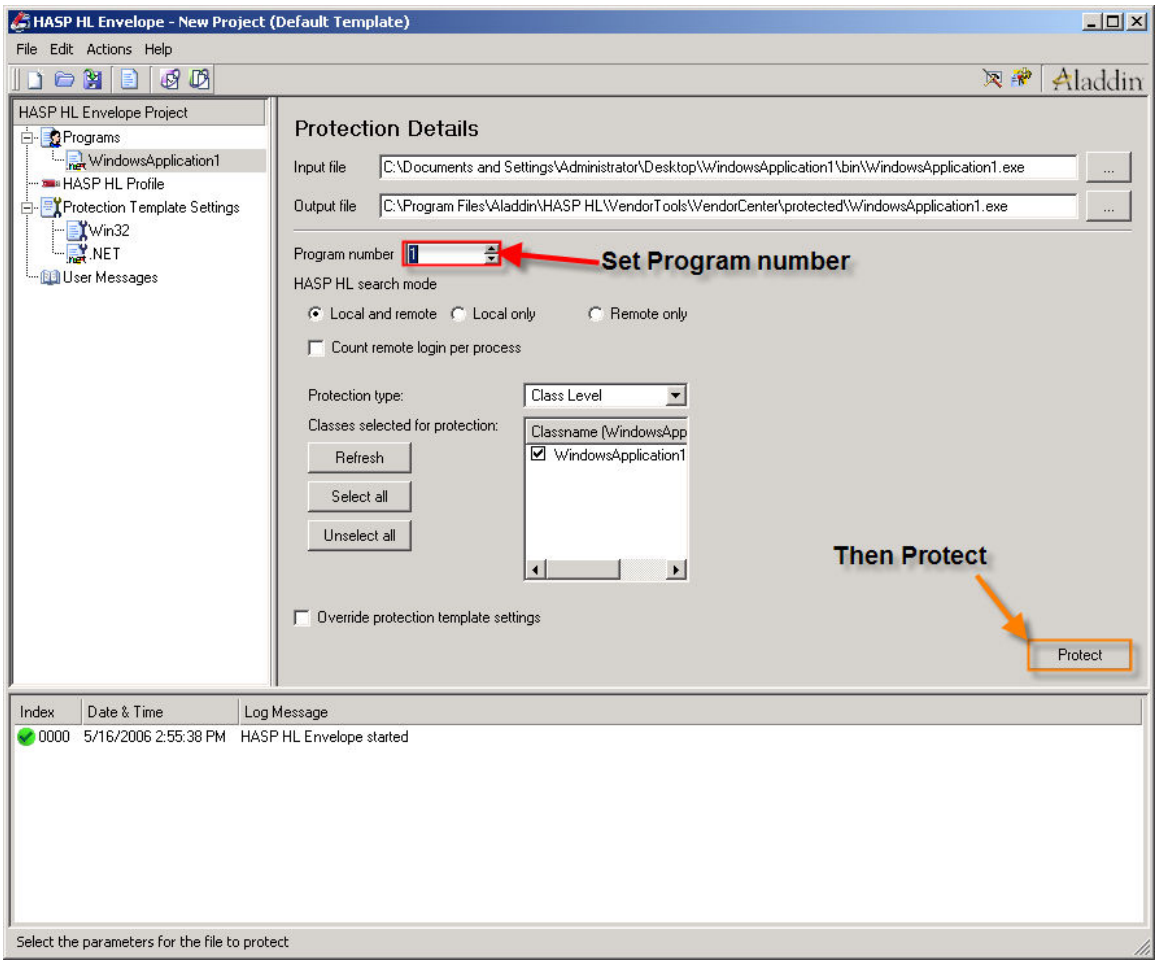


Figure 4.1.9 – Set the program number to an unused number, and click Protect to make the protected executable.

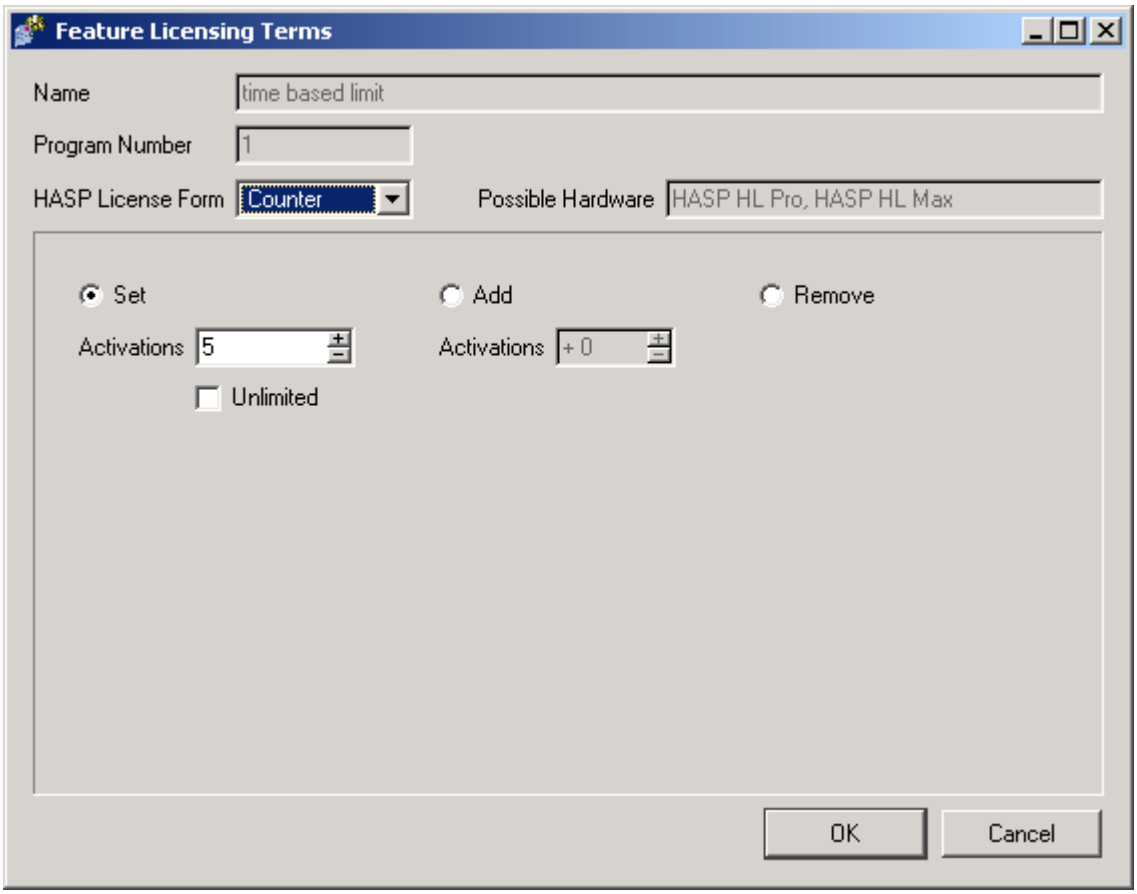


Figure 4.1.10 – Set the licensing model and terms.

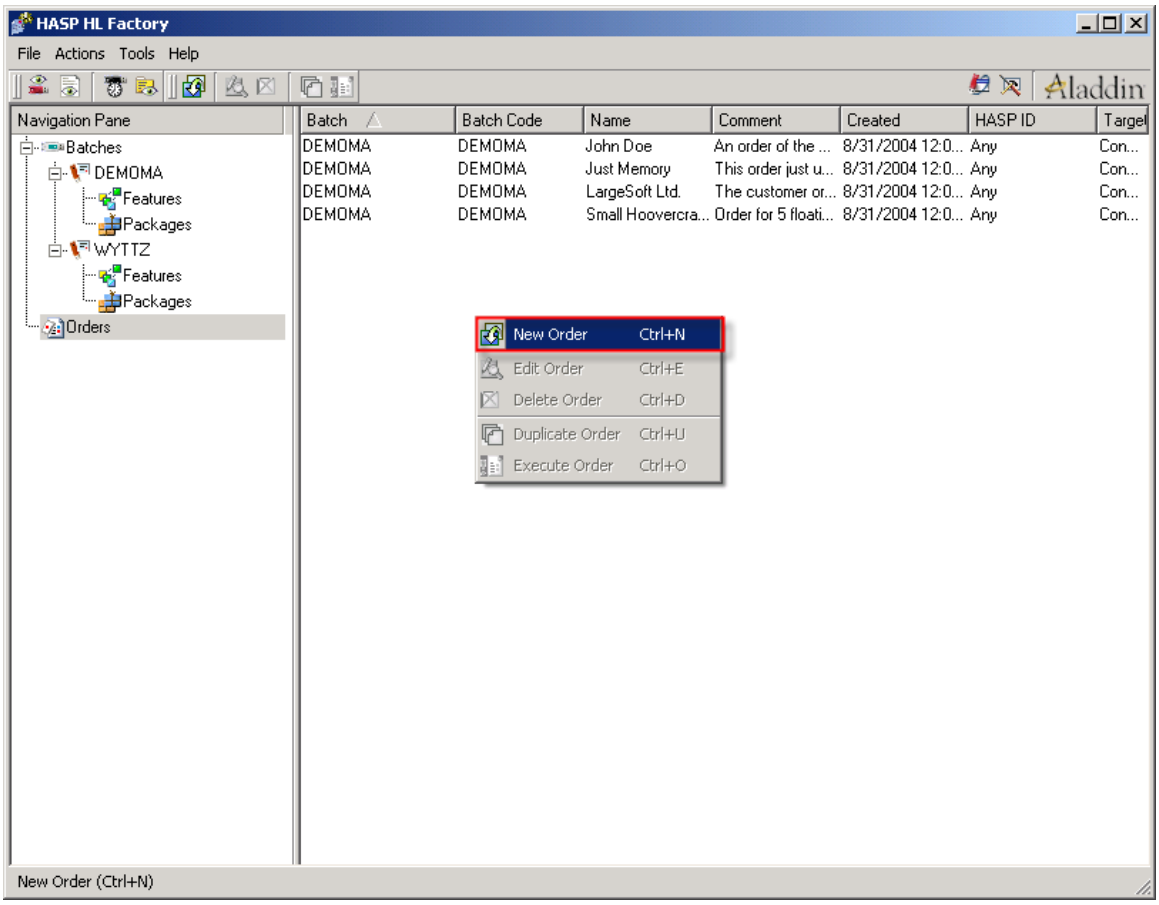


Figure 4.1.11 – Make a new order.

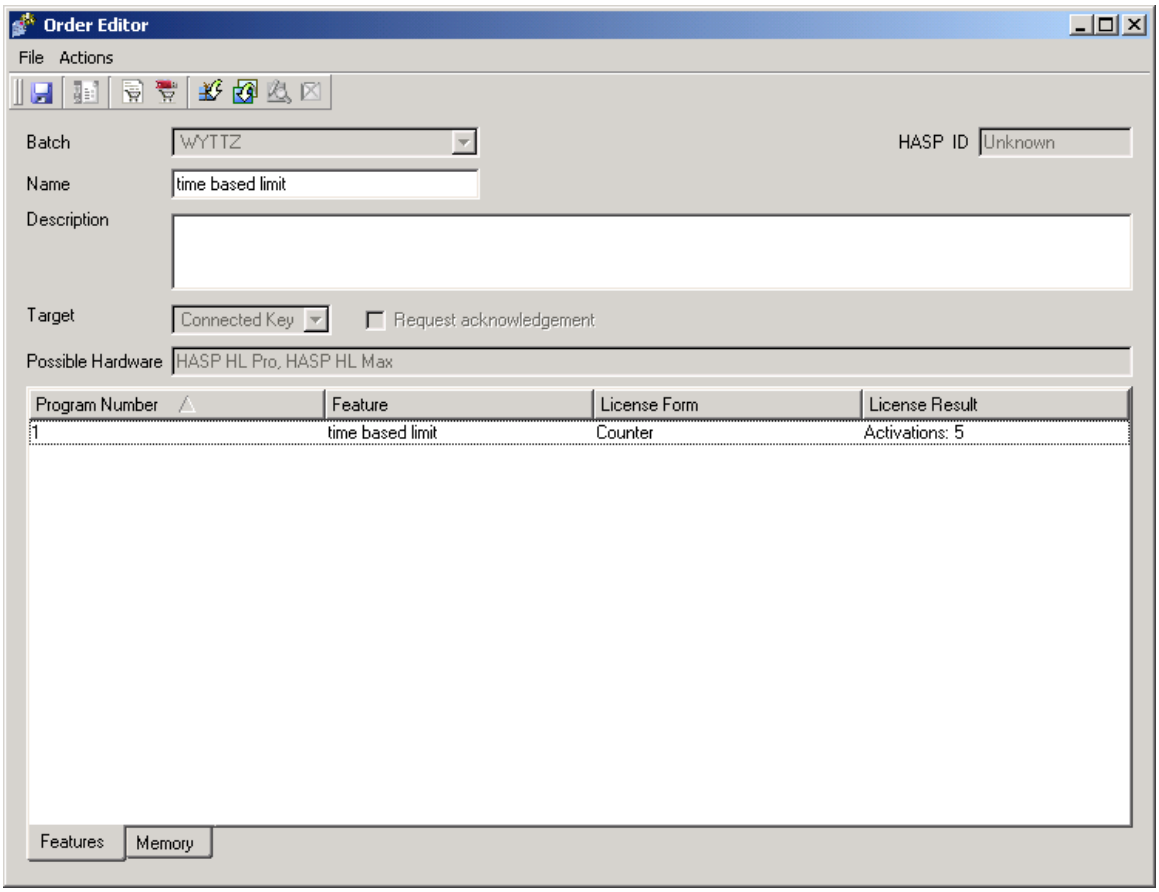


Figure 4.1.12 – Add the package that contains the features you created.

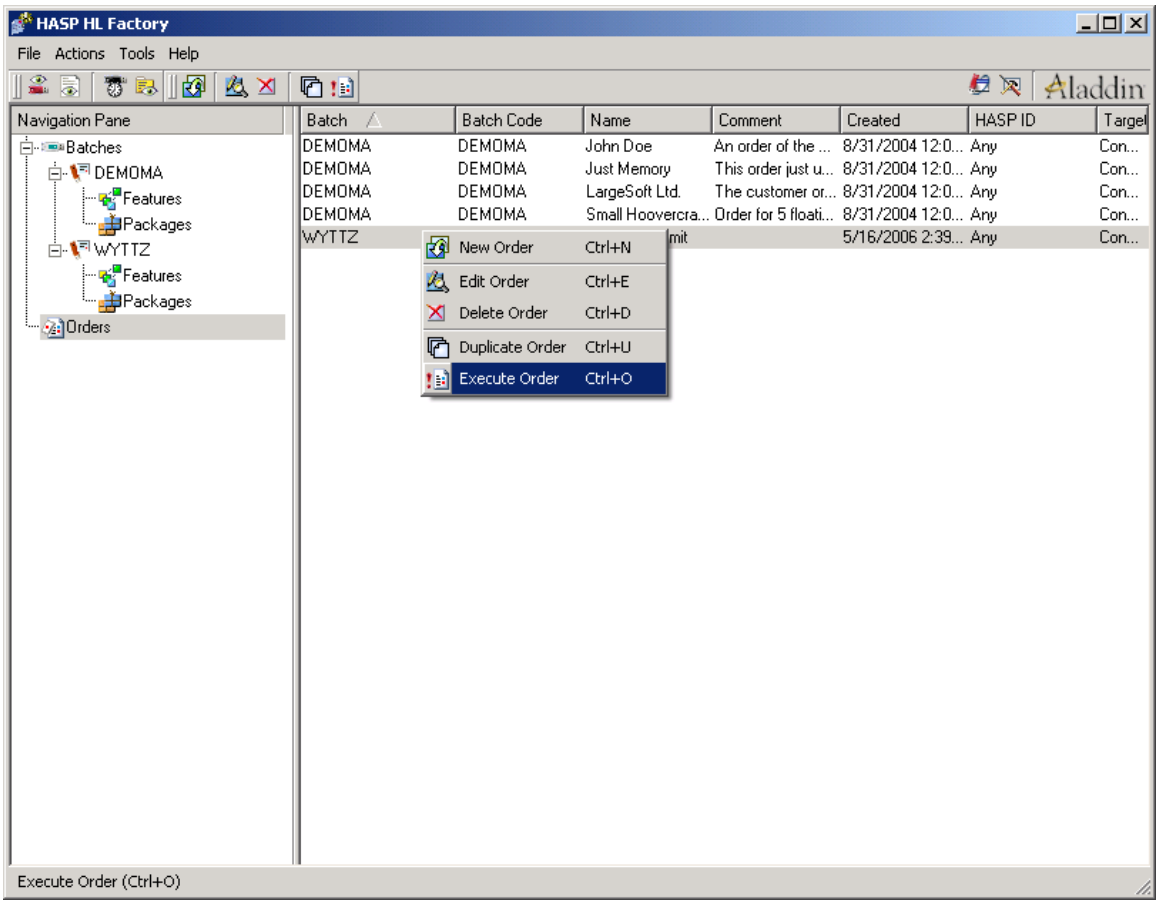


Figure 4.1.13 – Execute the order that you just created. This will write the license data to the user key.

Scoring

HASP 70
Sentinel 100

Scoring

Customer & Developer Perspective	Aladdin HASP HL	SafeNet Sentinel Hardware Key
Relative Advantage (2 tests)	100	100
Complexity (4 tests)	96	79.5
Compatibility (2 tests)	95	70
Observability (1 test)	90	80
Trialability (3 tests)	90	80
Overall	**94.2**	**81.9**

Relative Advantage – Using hardware token authentication as a licensing model for software applications has many advantages as well as disadvantages. Licenses are, in general, very different for these applications versus applications protected by other means (serial numbers, product activation, etc). The best choice can only be made by the purchasing user, as he must make the decision if a physical token with its licensing model of "where the key goes, the license goes" is a better choice for him instead of the typical "one install per machine per serial/activation" model of many other applications.

Complexity – Overall, the HASP package had a finer level of detail over the whole protection process than Sentinel. There were more steps required in using the HASP package; however, the whole experience felt more user-friendly. Sentinel had a shorter and easier process to wrap a shell around an existing program, but the experience of trying to integrate the Sentinel API into a program could be improved. For end-users of both products, after the protected applications and drivers were successfully installed, there was very little difference in running protected applications.

Compatibility – Both products performed in as expected on the test platform. The HASP uninstaller could be improved to remove all the files installed on the system, instead of leaving the four files the uninstaller failed to remove. When a protected Sentinel application was run on Windows 2003 Server, the program was able to query the key and launch the application. However, after the Terminal Services service was installed on the machine, the program reported it was unable to find the keys.

Observability – The HASP kit with which the experiments were preformed came with one developer key and five user keys. There were no issues in making a replacement key if the first key for a protected program was lost or broken. The Sentinel kit contained only one user key, and attempts to use a second user key from another Sentinel kit was met with an error saying that the developer/user key combination was invalid, making the test less complete that expected. Attempts to contact SafeNet through the proper channels to obtain better information about replacement keys were met with difficulty.

Trialability – HASP offered better phone support and quicker turnaround time for inquires about receiving a development kit. Sentinel was not able to return the calls made to them within the period of this experiment. Sentinel's hours are better suited to worldwide customers, and it appears their call centers are more diversified. HASP has a

more complicated structure of adding an expiring license to their protected programs, as well as poorer default error messages.

Section 7: Application Programming Interface

Introduction

In order to compare the Aladdin HASP HL and the SafeNet Sentinel, a set of expectations were developed before each unit was examined. Once these were developed, a scale was generated for use in scoring the two products. Some features sought in these products are language support, compiler support, consistency of implementation across supported languages, and API ease of use.

To compare the products offered by SafeNet and Aladdin fairly, a default test platform was chosen. The platform consisted of a default installation of Microsoft Windows XP with Service Pack 2 with every security update available as of April 28, 2006 applied.

Features

The SafeNet Sentinel hardware key supports three compilers: Microsoft compilers, Borland compilers, and Java compilers. This resulted in a fair amount of languages documented as supported. The SafeNet installer provides an option to "Browse the programming interfaces." Once this option is selected, a three-tiered window is displayed showing the option to view Microsoft Compilers (Static & Dynamic), Borland Compilers (Static & Dynamic), and Java Compilers (Dynamic). The languages supported by the aforementioned compilers are as follows: C, C# .NET, Borland Delphi, Visual Basic .NET, Visual Basic 6.0, Java, and Borland C++. Each language implementation has a similar set of method calls that interact with the USB hardware key. When using the SafeNet API, the team found that not only does it include support for 128-bit AES encryption, but it also includes support to read and write information to the key. Due to a recent update to the SafeNet product line, the only currently supported platform is Microsoft Windows XP.

The Aladdin HASP HL features the ability to run on the three most used platforms available to users and developers alike. These platforms include Mac OS X, Linux distributions including RedHat, SUSE, and Microsoft Windows 98 though Server 2003. The HASP HL uses Digital Rights Management software protection based on AES and RSA algorithms. On page 18 of the HASP HL *Software Protection and Licensing Guide,* it indicates to use the API when access to the source code is available and when customization of the protection system is desired. The HASP HL uses methods inserted into the source code to protect the software. These methods can be used to lock out parts of the application unless the key is inserted, as well as store vital information needed by the program.

Experiments to be performed

A default installation of each hardware USB key was performed to provide a standard testing base. A full comparison of the compiler support between the SafeNet Sentinel and the Aladdin HASP HL keys was performed. Basic compiler support testing was conducted on both the Aladdin HASP HL and the SafeNet Sentinel hardware keys. When the compiler-testing phase was complete, each language supported was tested for

levels of completeness and quality of implementation. All languages tested included a corresponding ease-of-use experiment. Once preliminary compiler and language testing was complete, each method called under the languages used was evaluated for consistency across languages.

Interpretation of results

For each experiment that was run, a complete comparison of the actual results was compared to what each company stated in its documentation. Each of the following experiments was rated with a scale from 0 to 100. Due to the natural dependencies built into the structure of the application-programming interface, when the experiments were completed an average was taken of all API scores.

Experiment 1: Language Support

Description

The language support experiments consisted of testing both SafeNet Sentinel and Aladdin HASP HL for supported programming languages.

Assertion

Language support by both of these products was sufficient. The base languages supported by the corresponding toolboxes were almost identical. The SafeNet API Toolbox had built-in code sketch generators for ANSI C, Borland Delphi, C#, Visual Basic, Visual Basic .NET. The Aladdin Toolbox supported C, C#, C++, and Visual Basic .NET for automatic code generation. Considering the languages that have the built-in code generation facilities, each product was arguably similar. Looking deeper into the supported languages, it was found that both Aladdin and SafeNet share the most common core languages. Aladdin had support for more obscure languages like 4D and AutoCAD. Excluding this, the Aladdin HASP HL and the SafeNet Sentinel Hardware Keys had similar language support.

Experiment

A multi-stage experiment was conducted to test the language support offered by the SafeNet Sentinel and the Aladdin HASP HL hardware keys effectively and thoroughly. The two main objectives in this experiment were to find the extent of the languages supported. In addition, the code samples supplied by each company were reviewed for quality.

The first stage of experimentation was to find the supported languages referenced in the documentation of the respective manufacturer's user-guide. After all the information from the user-guide was collected, a default install of both the SafeNet Sentinel API Toolbox and the Aladdin HASP HL API Toolbox was performed.

Analysis

The researchers collected information about supported languages from both Aladdin and SafeNet. The SafeNet Sentinel supported languages are listed in Figure 1.1.1 under supporting data, and the Aladdin HASP HL supported languages are listed in Figure 1.1.2. The data presented by both companies showed that the Aladdin HASP has more supported languages overall. It is worth noting that some additional languages supported by HASP are relatively obscure, such as 4D and AutoCAD.

Supporting Data

Borland C	C#	Delphi Dynamic
Visual C++	Visual Basic	Java 1.5
COM	Visual Basic .NET	Borland C++
Delphi Static		

1.1.1 – Languages supported by SafeNet Sentinel via document reference

4D	C#	AutoCAD
Delphi Dynamic	Delphi Static	FoxPro
Java	REAL Basic	Visual Basic .NET
C	Visual Basic	C++

1.1.2 – Languages supported by Aladdin HASP HL via document reference

C	C#.NET	Borland Delphi
Visual Basic	Visual Basic .NET	

1.2.1 – Languages supported by SafeNet Sentinel using a default installation running on the test platform

C	C++
C#.NET	Visual Basic .NET

1.2.2 – Languages support by Aladdin HASP HL using a default installation running on the test platform

Experiment 2: API Toolbox Ease-Of-Use

Description

The Application Programming Interface is where all the important functions for the developer reside. This is where the code sketches are generated for use in applications as well as testing functionality of the developer license models. A rating was given to both SafeNet and Aladdin, which was based on how easy it was to get source code with full functionality to have protected areas of code that would not run unless the hardware key was inserted.

Assertion

Both SafeNet Sentinel Hardware Keys and Aladdin HASP HL included sample code. However, many times the sample code for both products did not work even when instructions were followed to the letter. The researchers found the documentation to be insufficient in the areas of setup and linking the required dynamic link libraries. That being said, using the toolbox to test license models worked flawlessly. The researchers believe that implementing the actual hardware key into the program via the API method calls was not documented sufficiently or effectively.

Experiment

Multiple "hello world" applications were writing in a C# and Visual Basic .NET. Referring to the sample programs used in experiment 1, the simple "hello world" applications were chosen because of the simplicity of implementation and readability of the source code. The steps necessary to implement the hardware key into the code were observed and recorded. If a complete implementation of the hardware key was not possible in the time allotted for each application, then it was noted where the researcher had a problem and what may have been the cause.

Analysis

It was apparent that there was not sufficient documentation to complete this task effectively or efficiently during the initial phase of this experiment. All of the researchers understood how to use the API from the documentation, but a consensus was reached among the researchers that insufficient documentation existed to merge the API functions into the application. The sample code offered did not run successfully or exited prematurely. These errors included:

- C# and Visual Basic .NET both returned the same error when trying to reference the .dll file in the project when using the SafeNet API.
- The C API had a link failure when trying to link both dynamically and statically when using the SafeNet API.
- The researchers found the Java implementation to be difficult to use because supplied documentation was not sufficient for both the SafeNet API and the Aladdin API.

The actual toolbox for the SafeNet Sentinel was easier to navigate and presented more information to the user.

Experiment 3: Method Consistency between Languages

Description

The Application Programming Interface is like a toolbox for all the important functions developers use. The supplied tools from each vendor provided the ability to create code sketches. These sketches can be used in applications. There is also an interface for testing the functionality of the developer's license models. A rating was given to both SafeNet and Aladdin based upon how easy it was to get source code with full functionality to protect areas of code.

Assertion

Both the Aladdin HASP HL and the SafeNet Sentinel API's were functionally equivalent between the supported languages.

Experiment

For the experiment, we chose to view the code sketch to query the key generated by the Aladdin HASL Toolbox and the SafeNet Sentinel Toolbox respectively. After the code sketches were generated, they were compared to each other. The input required and the output generated was tested and reviewed to ensure functional equivalence between the supported languages.

Analysis

Code consistency is very important when developing multiple applications in different languages. Learning an API once and being able to implement the API quickly and easily saves time and money. According to the team's research (figures 3.1.1 – 3.1.3), it is apparent that SafeNet has a very strict standard when creating the code sketches from the toolbox. In each of the sketches provided, there are 4 variables with the equivalent type of an unsigned int, then the required GetLicense() method. The Aladdin HASP has some discrepancies between languages that may be worth mentioning. According to our test, the result is the same but errors are handled differently when using the C API. If the C API is being used, each method is handled on a case-by-case basis.

Supporting Data

```
unsigned long int   status;
unsigned long int   DeveloperID;
unsigned long int   licID;
unsigned long int   flags;

status = SFNTGetLicense( DeveloperID,
                         SOFTWARE_KEY,
                         licID,
                         flags,
                         &licHandle );
```

3.1.1 - ANSI C SafeNet GetLicense Method Call

```
uint    status;
uint    DeveloperID;
uint    licID;
uint    flags;

status = sentinelkey.SFNTGetLicense(    DeveloperID,
                                        SentinelKeysLicense.SOFTWARE_KEY,
                                        licID,
                                         flags);
```

3.1.2 - C# SafeNet GetLicense Method Call

```
Dim   status       As Int32
Dim   DeveloperID  As Int32
Dim   licID        As Int32
Dim   flags        As Int32

status = sentinelkey.SFNTGetLicense(    DeveloperID, _
                                        sentinelKeysLicense.SOFTWARE_KEY, _
                                        licID, _
                                         flags )
```

3.1.3 - Visual Basic .NET SafeNet GetLicense Method Call

```
const hasp_feature_t feature = HASP_PROGNUM_DEFAULT_FID;

hasp_handle_t handle = 0;
hasp_status_t status;

unsigned char vendor_code[] = "vendor code goes here";

status = hasp_login(feature, vendor_code, &handle);

/* check if operation was successful */
if (status != HASP_STATUS_OK)
{
   switch (status)
   {
      case HASP_FEATURE_NOT_FOUND:
         break;
      case HASP_CONTAINER_NOT_FOUND:
         break;
      case HASP_OLD_DRIVER:
         break;
      case HASP_NO_DRIVER:
         break;
      case HASP_INV_VCODE:
         break;
      case HASP_FEATURE_TYPE_NOT_IMPL:
         break;
      case HASP_TMOF:
         break;
      case HASP_TS_DETECTED:
         break;
      default:
         break;
   }
}
```

4.2.1 - HASP Login Method in C

```
HASPFeature feature = HASPFeature.ProgNumDefault;

string vendorCode = "vendor code goes here";

HASP hasp = new HASP(feature);
byte[] code = ASCIIEncoding.Default.GetBytes(vendorCode);
HASPStatus status = hasp.Login(code);

if (HASPStatus.StatusOk != status)
{
   //handle error
}
```

4.2.2 - HASP Login Method in C#

```
Dim feature As HASPFeature = HASPFeature.ProgNumDefault

Dim vendorCode As String = _"vendor code goes here"

Dim hasp As HASP = new HASP(feature)
Dim status As HASPStatus = hasp.Login(UTF8Encoding.Default.GetBytes(vendorCode))

If (HASPStatus.StatusOk <> status) Then
  'handle error
End If
```

4.2.3 - HASP Login Method in Visual Basic .NET

Summary

The researchers believe that both of the application programming interfaces have room for improvement in their documentation. This assertion is based on the difficulty experienced with getting the application-programming interface implemented into the previous programming projects.

Scoring

API	**Aladdin** HASP HL	**SafeNet** Sentinel Hardware Key
Language Support	100	80
API Toolbox Ease-of-Use	50	60
Method Consistency	75	100
Overall	**75**	**80**

Language Support

The languages supported by both products were sufficient. The Aladdin HASP has more documented support languages, earning it a higher score.

API Toolbox Ease-of-Use

The researchers found both products to be hard to use due to lack of documentation. Many of the vendor-supplied samples were unable to execute within the test environment. From the point of view of a developer who may have never used such an API, both products may be intimidating. Improving the documentation, or providing a systematic guide similar to provided shell documentation, would improve the experience.

Method Consistency between Languages

For the SafeNet Sentinel, every language has nearly identical implementation code. Therefore, if the developer knows the syntax to a supported language he can have a working method by looking at the code sketch. The Aladdin HASP does not provide the same level of compatibility. With a C implementation, more information supplied to the user, such as type of status error if not successful, would be helpful. Other languages provided a *success* or *no success* variable.

References

Aladdin - Support & Downloads. 15 May 2006

<http://aladdin.com/support/hasp/vendor.asp>.

SafeNet-Inc - Hardware Keys. 15 May 2006

<http://safenet-inc.com/products/sentinel/hardware_keys.asp>.

Section 8: Vendor Tools

Introduction

There are graphical tools supplied to developers in every development kit from SafeNet and Aladdin. These vendor tools should be versatile enough to implement license models in almost every type of application the developer creates. Some of the major factors in usability of the vendor tools are customization of the license models, vendor tools user interface, and quantity of sample code.

To compare the products offered by SafeNet and Aladdin fairly, a default test platform was chosen. The platform consisted of a default installation of Microsoft Windows XP with Service Pack 2 with every security update available as of April 28, 2006 applied.

Features

The features that the SafeNet Sentinel and Aladdin HASP HL provide are extremely important to a developer for ease of implementation and protection. SafeNet Sentinel features a fast and easy way to protect an already compiled program by using its "Quick Shell" function. This uses popular pre-made license modules to "wrap" a program not allowing it to run unless the correct license is found on the key. The SafeNet vendor tools contain an "API Explorer" that allows the developer to experiment with the business layer API before adding the source code to the project. It also creates source code for use in programs in a supported programming language. The "Key Status Panel" displays information regarding the keys that are currently attached to the computer. In the panel, it shows the key type, the device ID, and serial number. It also allows one to choose which key the developer would like to program.

The Aladdin HASP HL features the ability of "Automatic Licensing." This ability allows the software to check which modules are licensed to run, for how long, how many times, and how many users, without developer intervention, etcetera. The Aladdin HASP HL vendor tools package contains the "HASP HL Envelope," the "HASP HL Factory," and the "HASP HL Toolbox." The HASP HL Envelope is used to apply a secure protective shield to the program. It is used to protect an application without having the source code. The HASP HL Factory is used to define and produce licenses for protecting an application as well as writing to the card's available memory. The HASP HL Toolbox is where the developer can learn to use the HASP HL API to generate code for use in protecting software by adding the code to the source.

Experiments to be performed

To develop a process for comparing the vendor tools ease-of-use, the quality of code samples, and developer tools, three tests were created. Each test explored the way that the toolkit was built for ease-of-use, how extensively the toolkit explained what each function did, and how well the toolkit generated source code.

Experiment 1: User Interface

Description

Visibility and simplicity are key factors in the developers' interactions with a product's vendor tools. To test the SafeNet Sentinel and the Aladdin HASP HL's user interface, a generalized set of questions needed to be answered using each respective vendor tools. These questions included how easy it was to locate each products features, the uniformity of each part, and the overall simplicity.

Details

Rather than testing the capability of the user interface, this experiment addressed how the capability was presented to the user of the toolkit. Figure 1 through Figure 13 show each accessible window, starting from installation of the vendor tools to using it to protect applications. These are not all of the sections available within the vendor tools, but these were presented as the most important and valuable parts of each vendor's software.

When started, the SafeNet Sentinel's toolkit brought up a main window that contained the ability to open the actual vendor toolkit and the manuals, show the interfaces, and install the drivers for the keys. Since this experiment pertains to the vendor toolkit, the ability to view the interfaces lacked information on what to do once inside the files. The researchers have yet to implement SafeNet Sentinel protection in Java-based software due to the lack of specific directions regarding that task. Once in the toolkit, a main screen (Figure 10) indicated its features at the left side. The frame-based system that Sentinel uses to display each feature is highly valuable. It provides a combination of ease of use, simplicity, and visibility. While on one feature, the developer can view all other features without having to go back or close the window currently in use. Figures 10 through 13 show each feature available once inside the toolkit. Although each feature was easily visible, it was required to return to the first window (Figure 9) to view the manuals. This made quick reference more difficult should a problem arise. Although the manuals were more difficult to reach, the API Explorer allowed easy and simple access to the methods and functions available to the developer. The SafeNet Sentinel did well with usability and visibility, but it was less simple for quick updates or changes. If manuals were available from inside the vendor tools, the SafeNet Sentinel toolkit would be more efficient.

The Aladdin HASP HL vendor toolkit performed well with regard to simplicity and visibility. When started, the toolkit displayed the protection features available to the developer, as well as tutorials, and an updater (Figure 1). Once a feature was selected, it opened in a new window (Figures 4, 5, 6). The HASP HL Envelope function opened in a new window the protection feature that used a secure shell as the way to protect the application. Each option selected in the main window (Figure 4) opened that function in a new window. The fact that a new window was created with each click in the main window could be either good or bad depending on the opinion of the developer. The amount of open windows could be considered a hindrance. Overall, the Aladdin HASP HL toolkit succeeded in simply and easily giving the developer all of the resources required to implement the Aladdin protection features into an application.

Supporting Data

Aladdin HASP HL Screenshots

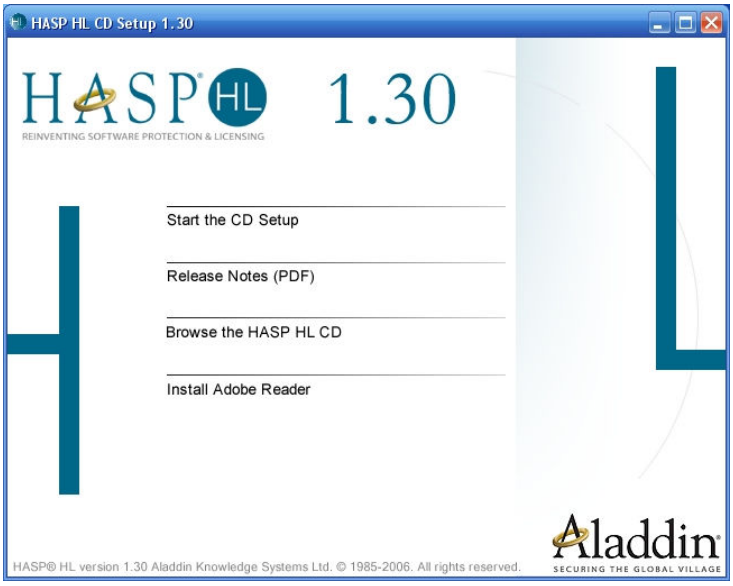

HASP HL CD Setup 1.30
HASP HL
1.30
REINVENTING SOFTWARE PROTECTION & LICENSING
Start the CD Setup
Release Notes (PDF)
Browse the HASP HL CD
Install Adobe Reader
Aladdin
SECURING THE GLOBAL VILLAGE
HASP® HL version 1.30 Aladdin Knowledge Systems Ltd. © 1985-2006. All rights reserved.

Figure 1: Auto run screen

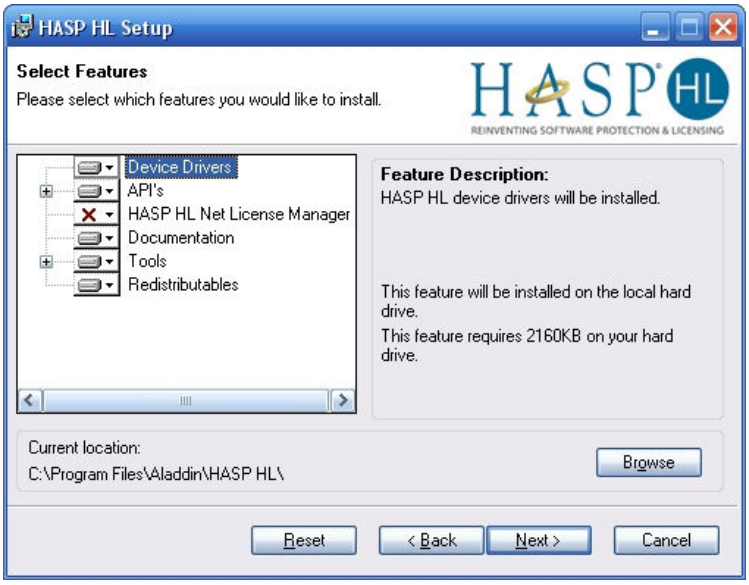


Figure 2: Advanced installation options

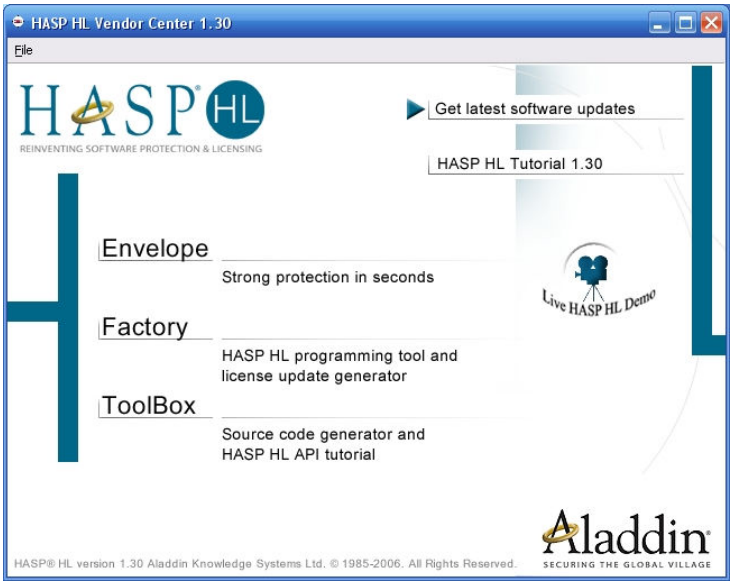


Figure 3: Main screen

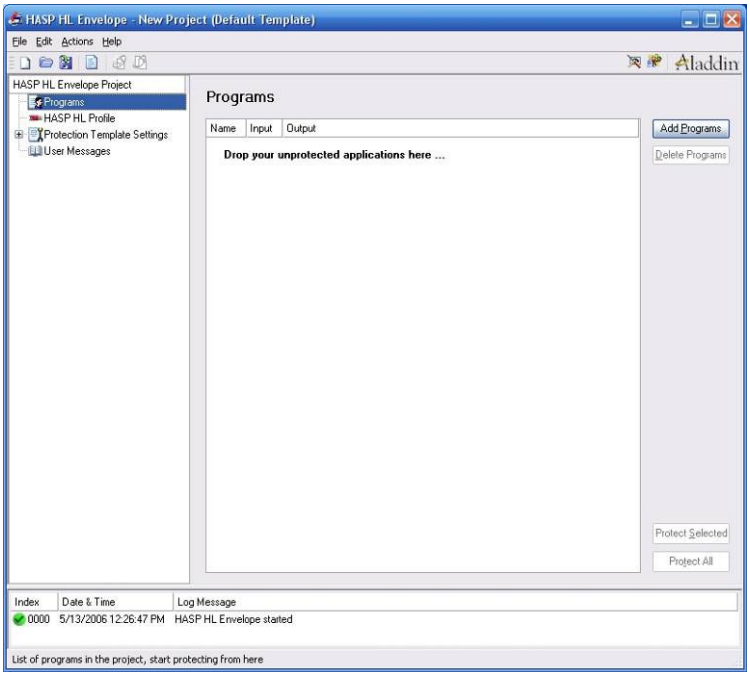


Figure 4: Envelope protection project creator

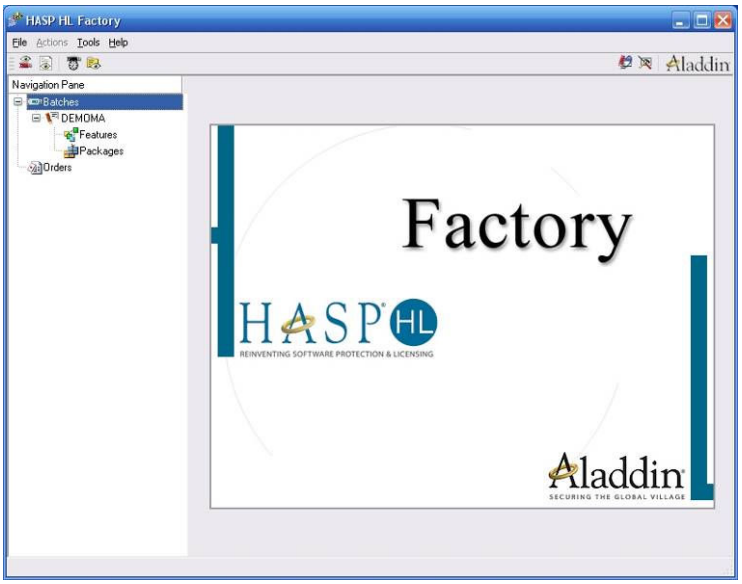


Figure 5: Factory protection area

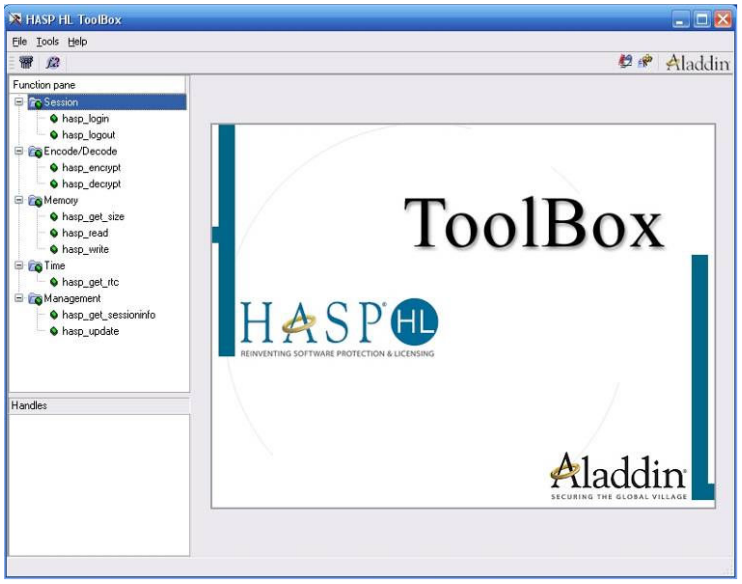


Figure 6: Toolbox protection area

SafeNet Sentinel Screenshots

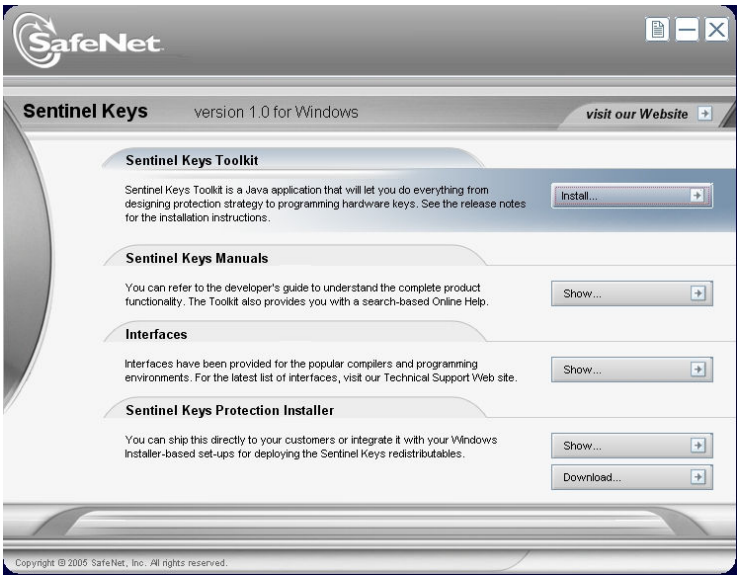

SafeNet
Sentinel Keys
version 1.0 for Windows
visit our Website
Sentinel Keys Toolkit
Sentinel Keys Toolkit is a Java application that will let you do everything from designing protection strategy to programming hardware keys. See the release notes for the installation instructions.
Install...
Sentinel Keys Manuals
You can refer to the developer's guide to understand the complete product functionality. The Toolkit also provides you with a search-based Online Help.
Show...
Interfaces
Interfaces have been provided for the popular compilers and programming environments. For the latest list of interfaces, visit our Technical Support Web site.
Show...
Sentinel Keys Protection Installer
You can ship this directly to your customers or integrate it with your Windows Installer-based set-ups for deploying the Sentinel Keys redistributables.
Show...
Download...
Copyright © 2005 SafeNet, Inc. All rights reserved.

Figure 7: Auto run screen

Sentinel Keys 1.0.0 (English) - InstallShield Wizard
Custom Setup
Select the program features you want installed.
Sentinel
Click on an icon in the list below to change how a feature is installed.
Sentinel Keys Driver
Compiler Interfaces
Manuals
Secure Update module
Configuration File
Sentinel Keys Protection Installer
Sentinel Keys Server
Sentinel Keys Toolkit
Feature Description
Installs all the items given below.
This feature requires 0KB on your hard drive. It has 1 of 1 subfeatures selected. The subfeatures require 25KB on your hard drive.
Install to:
C:\Program Files\SafeNet Sentinel\Sentinel Keys\1.0\
InstallShield
Help
Space
< Back
Next >
Cancel

Figure 8: Advanced installation options

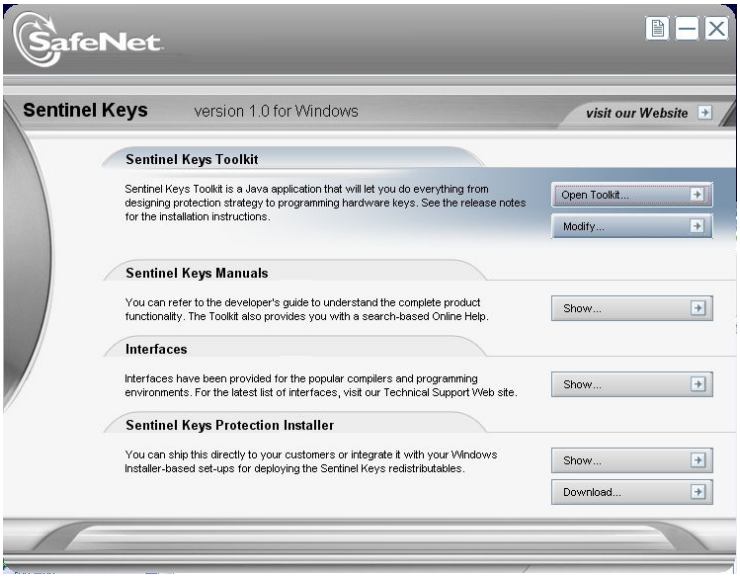

SafeNet
Sentinel Keys
version 1.0 for Windows
visit our Website
Sentinel Keys Toolkit
Sentinel Keys Toolkit is a Java application that will let you do everything from designing protection strategy to programming hardware keys. See the release notes for the installation instructions.
Open Toolkit...
Modify...
Sentinel Keys Manuals
You can refer to the developer's guide to understand the complete product functionality. The Toolkit also provides you with a search-based Online Help.
Show...
Interfaces
Interfaces have been provided for the popular compilers and programming environments. For the latest list of interfaces, visit our Technical Support Web site.
Show...
Sentinel Keys Protection Installer
You can ship this directly to your customers or integrate it with your Windows Installer-based set-ups for deploying the Sentinel Keys redistributables.
Show...
Download...

Figure 9: Main screen

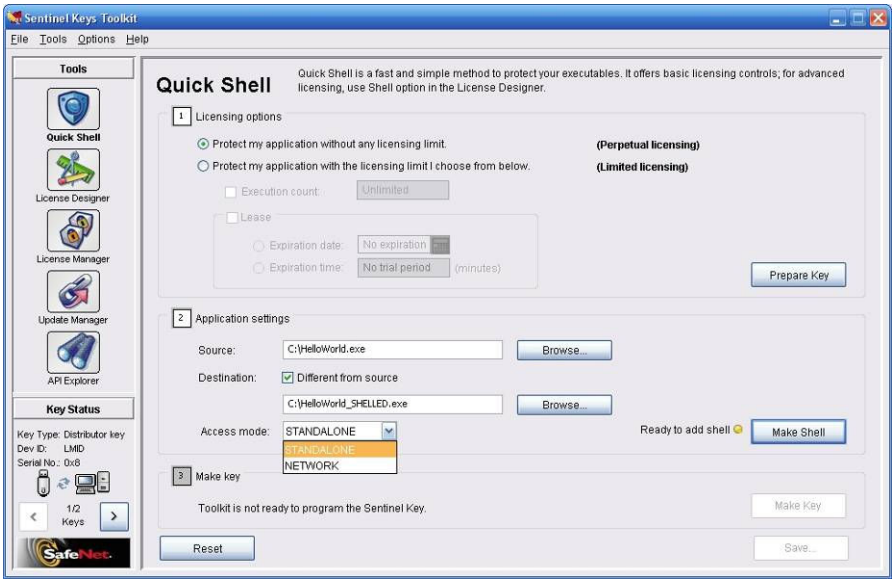


Figure 10: Quick shell protection area

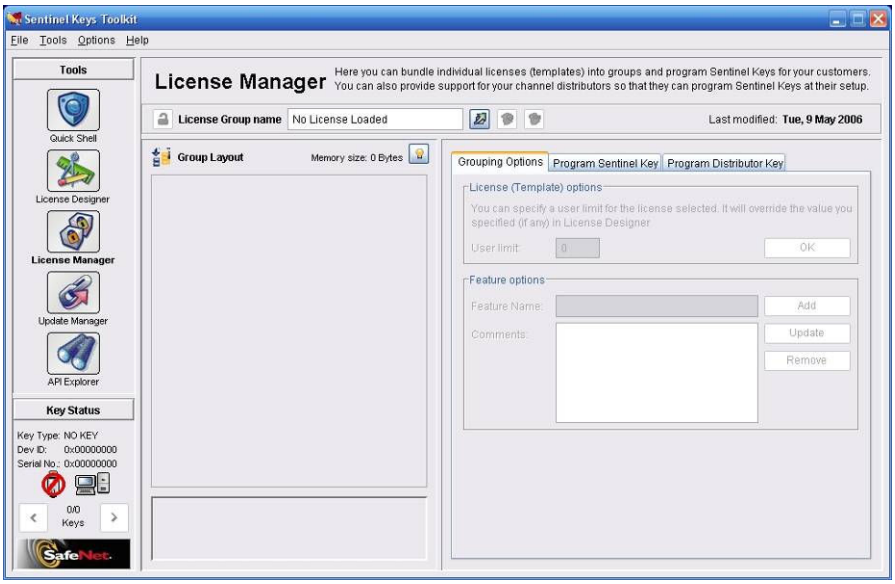


Figure 11: License manager

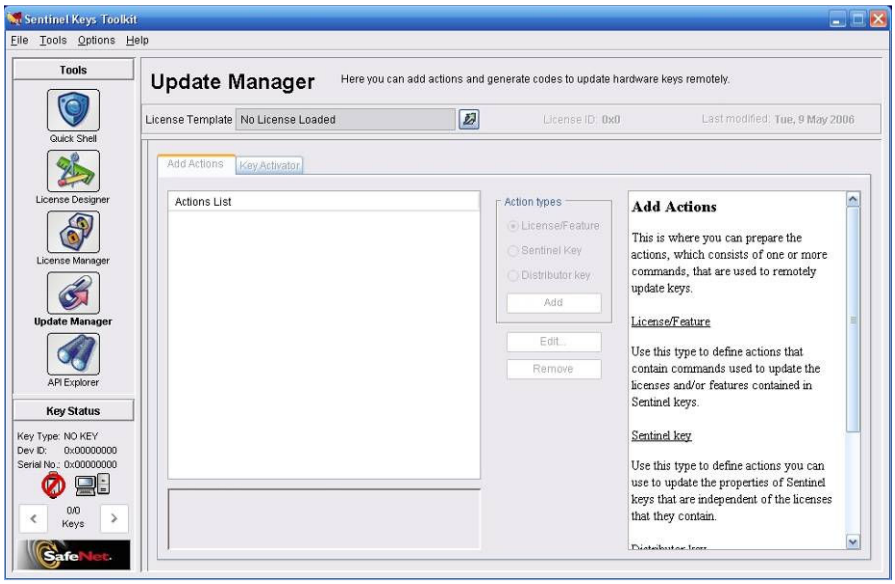

Sentinel Keys Toolkit
File Tools Options Help
Tools
Quick Shell
License Designer
License Manager
Update Manager
API Explorer
Key Status
Key Type: NO KEY
Dev ID: 0x00000000
Serial No.: 0x00000000
0/0
Keys
SafeNet.
Update Manager
Here you can add actions and generate codes to update hardware keys remotely.
License Template
No License Loaded
License ID: 0x0
Last modified: Tue, 9 May 2006
Add Actions
Key Activator
Actions List
Action types
License/Feature
Sentinel Key
Distributor key
Add
Edit...
Remove
Add Actions
This is where you can prepare the actions, which consists of one or more commands, that are used to remotely update keys.
License/Feature
Use this type to define actions that contain commands used to update the licenses and/or features contained in Sentinel keys.
Sentinel key
Use this type to define actions you can use to update the properties of Sentinel keys that are independent of the licenses that they contain.

Figure 12: Update manager

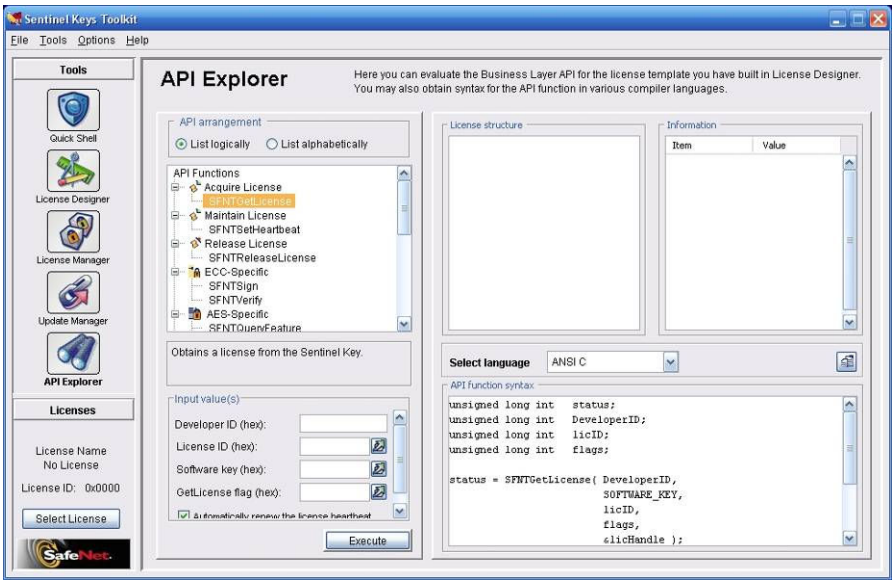


Figure 13: API Explorer

Experiment 2: Toolbox Functionality

Description

Toolbox functionality is an important part of a piece of software. The appearance of the software means little if it does not function well to complete the tasks required. Testing the functionality of the Aladdin HASP HL vendor toolkit and the SafeNet Sentinel vendor toolkit was based upon questions that were asked while the experiments were taking place. The first question asked about the simplicity of each task, which included how many steps were required to do a task. The second question asked whether the toolbox gave the user options throughout each step. This included customization of each level of installation and eventually each level of protection.

Details

The first experiment tested how many steps and decisions were required to shell an application with the provided license settings in each vendor's toolkit.

On a test platform that was freshly restarted, the Sentinel SafeNet toolkit took five steps to create a shelled application using a predefined license. While using a local access mode, there was an option to use a network access mode. The licensing options also allowed a limited execution count or time based lease. After the experiments concluded, it was determined that the SafeNet Sentinel "Quick Shell" was a simple but powerful tool for securing an application. It allowed a quick design of a license as well as the default unlimited perpetual license. It added greatly to the overall functionality of the toolkit (Figure 1).

Once again using a freshly restarted test base, the Aladdin HASP HL toolkit took two steps to create a shelled application. The first step selected which application the developer wanted to protect and the second step was to secure it. With the ability to create a secured application in a very few clicks, the Aladdin HASP HL's "Envelope"

was extremely functional. Not only did it quickly do the task the developer required of it, but it also gave an abundance of options (Figure 2).

The second part of this experiment tested the functionality of the SafeNet Sentinel and the Aladdin HASP HL toolkits. It explored options available to create a license template as well as the ease of implementation of that same license.

When the SafeNet Sentinel "License Designer" is selected, the "License Designer Wizard" starts. The first option asks the developer if the user was inexperienced with features or templates and if he would like to create a new one. The second option asks if the developer would like to copy a sample template and save it as a new license. The final option available would add features under Shell/API tabs. To experiment this process fully, the researchers chose to create a new license. The second option allows the developer to select whether to use a shell feature or an API feature. Choosing to create a shell feature prompts the user to supply a binary or data file. Once selected, a destination path was chosen. The shell attributes then were chosen. The attributes were active, lease, and limit exceptions. Since the research team did not draw up a license design prior to the start of testing, the active attribute was selected. If a different attribute is selected, more options would be available, such as expiration date and time, and the execution count. The next decision was to indicate the license network options – these included standalone and network. If network is selected, there is an option to allow license sharing and the time to maintain a license. On the same screen, advanced options were available. These options were multi-layer leveled with choices 1 through 5, the ability to run the program with a debugger present, to tell the program that Shell SDK was utilized, and to hide or show import symbols. The last step asked for the name of the license that was created, the owner, the comments, and the license constraints in user limit.

The SafeNet Sentinel "License Designer" then showed the license that was just created, allowing the developer to edit or secure an application with it. The "License Designer" functions well, it allows a developer who may not be experienced with the operation to create a license and use it without many difficult decisions. Throughout the creation of the license, the left side of the window indicated how many steps were left, and how many were completed. The SafeNet Sentinel's license creation and management application simply and successfully created and secured an application in seven steps.

The Aladdin HASP HL Factory is the application that gives the developer the ability to create, edit, or update a license. The way that it creates licenses and edits them forces the information to fall under the API part of this document. Although its functions and data fall under the API part of this document, the researchers found that it was very complicated to use and hard to understand, even with the assistance of the manual. Please refer to the API section of this document to read about the Aladdin HASP HL's Factory.

Supporting Data

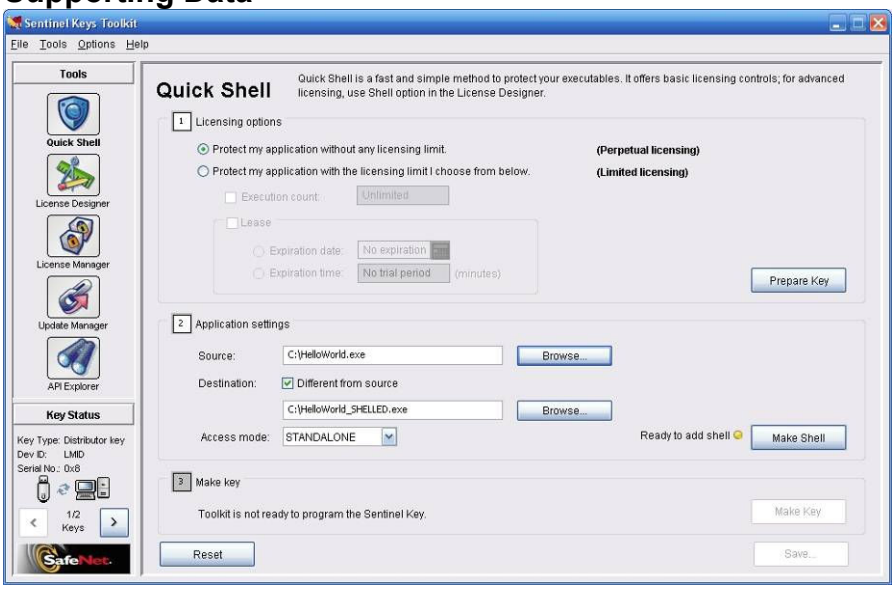


Figure 1: The SafeNet Sentinel "Quick Shell" application

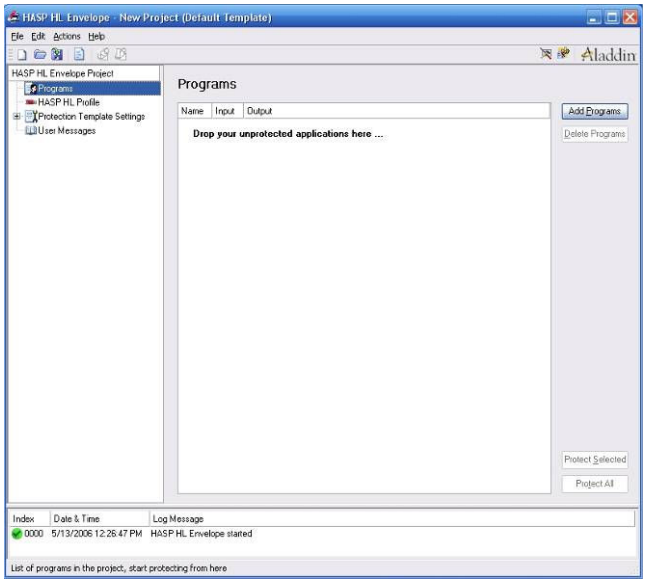


Figure 2: The Aladdin HASP HL "Envelope" application

Experiment 3: Code Generation Facilities

Description

Generated code is a very important feature in a vendor toolkit. The ability to generate working code makes the task of protecting an application easier. The term *generate* was used loosely; the ability of the program to actually generate code differed, but both did, in fact. The ability to customize each API call, therefore, was the common test ground. The facilities in these experiments were tested for simplicity, quality, and customization. Customization involved the available languages its code could generate and the

expandability of the library of languages. The researchers used C as the language to test the functionality of the test language because it is a default language for both toolkits.

Details

The SafeNet Sentinel API Explorer is a tool that SafeNet has supplied to the developer to generate code and view the API method calls. Once the API Explorer was selected on the left, the main frame showed the method calls along with the license structure, information, and the API function syntax. Once the developer has changed the calls to what the license required, it could be executed and the key was updated to use the new settings.

The Aladdin HASP HL Toolbox is what the vendor supplied the developer for viewing and generating code. The functions available for viewing and editing are located on the left side. When a function is selected (like encryption function), one is given the generated code at the bottom. Along with the code at the bottom, one is given an option whose range depends on the function. The encrypt function has the ability to change the size, load a new file, reset the buffer, or save as. If a change is made, one is given the option to execute the new code, which allows one to use it with a program because the application has updated the key to the setting the developer has changed.

Supporting Data

```
unsigned long int  status;
unsigned long int  featureID;
unsigned char plainBuffer[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
unsigned char cipherBuffer[16];

status = SFNTEncrypt( licHandle,
                      featureID,
                      plainBuffer,
                      cipherBuffer);

if (status != SP_SUCCESS) {
  // If featureID is invalid, then this API will return error.
}
```

Figure 1: SFNTEncrypt API function in C

```
hasp_size_t len = 16;

hasp_status_t status;

unsigned char data[] = {
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

status = hasp_encrypt(handle, data, len);

/* check if operation was successful */
if (status != HASP_STATUS_OK)
{
   switch (status)
   {
      case HASP_FEATURE_NOT_FOUND:
         break;
      case HASP_INV_HND:
         break;
      case HASP_TOO_SHORT:
         break;
      case HASP_ENC_NOT_SUPP:
         break;
      default:
         break;
   }
}
```

Figure 2: hasp_encrypt API function in C

Experiment 4: Sample Code Quality

Description

To implement protection architectures correctly and securely, a developer does not want to attempt this without an example of how it should be done. To do this, a developer will seek sample code to see ways in which it can be done. If this code is not well written, a developer may do something incorrectly, causing a serious risk to the software's stability or jeopardizing the protection of that piece of software. The quality of the sample code was tested by looking at the standards set for that programming language and determining how easily it would be understood by a developer attempting to use it.

Details

To test this, the researchers viewed the sample code supplied by the Aladdin HASP HL default install and the SafeNet Sentinel default install. The code from each was reviewed to determine whether it was readable, and whether it explained what each part of the code

did. Using C# to show comparable source code, each vendor gave a small example program. Looking through the code, the team decided that the SafeNet Sentinel and Aladdin HASP HL code was very similar in commenting and readability. Both followed closely the coding standards that were set by the programming language developers.

Supporting Data

```
/*********************************************************************/
/*                                                                   */
/*      Copyright (C) 2005 SafeNet, Inc.                             */
/*                       All Rights Reserved                         */
/*                                                                   */
/*********************************************************************/

/*C#.NET*************************************************************
* FILENAME    : LeaseDemo.cs
* DESCRIPTION :
*               Simple demonstration of Sentinel Key 1.0.0 library calls.
*
*             This is AES ALGO application. This application
*             will either run till 10 days from the date of first execution
*             or 10 times, whichever comes first. This program assumes that
*             an AES has been programmed into the key by Sentinel Keys
toolkit.
*
*C#.NET*/

using System;

namespace LeaseDemo
{
      /// <summary>
      /// Summary description for LeaseDemo
      /// </summary>
      public class LeaseDemo
      {
```

Figure 1: SafeNet Sentinel sample code

```
//////////////////////////////////////////////////////////////////////
//                 Aladdin Knowledge Systems (Deutschland) GmbH
//                     (c) AKS Ltd., All rights reserved
//
//
// $Id: hasp_demo.cs,v 1.2 2004/04/06 10:24:59 dieter Exp $
//////////////////////////////////////////////////////////////////////
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Text;
using Aladdin.HASP;


namespace HaspDemo
{
    /// <summary>
    /// HASP HL Sample Form
    /// </summary>
    public class FormHaspDemo : System.Windows.Forms.Form
    {
```

Figure 2: Aladdin HASP HL sample code

Analysis

The SafeNet Sentinel vendor toolkit and the Aladdin HASP HL vendor toolkit were comparable in function. Each had a set group of functions that allowed the developer to complete a task that would be required to protect an application. Throughout the test, a group of questions had been asked. Each answer would develop the outcome and results of this section of the document. Each experiment was worth 100 points, and the final score was the average of the four. Since each experiment dealt with something extremely relevant to developers, each experiment was weighted equally.

Scoring

Vendor Tools	**Aladdin** HASP HL	**SafeNet** Sentinel Hardware Key
User Interface	75	85
Toolbox Functionality	85	85
Code Generation Facilities	85	85
Sample Code Quality	90	90
Overall	**82.75**	**86.25**

References

Aladdin - Support & Downloads. 15 May 2006.

SafeNet-Inc - Hardware Keys. 15 May 2006.

Section 9: Security

Introduction

This section deals with the security implementation. The areas of focus are an overall look at how each USB key communicates with the licensed application, as well as the strength of the wrapping applied by each vendor.

Features

The SafeNet Sentinel uses a secure tunnel for communication to take place. This tunnel is created by using 128-bit AES (Advanced Encryption Standard) encryption and allows for secure communication between the USB Key and the client. When the developer selects an application to protect, this application is digitally signed by the developer key. It is signed using a 163-bit ECC (Elliptic Curve Cryptography) key that is contained within the developer key. This process is known as ECDSA or Elliptic Curve Digital Signature Algorithm.

On top of this, the developer can choose to use the API functionality to include the 163-bit ECC to generate session keys that will be used to encrypt data that is being transferred between the program and User Key.

SafeNet's implementation of program protection, as they refer to it as a "shell," consists of encrypting the program using 128-bit AES encryption as well as "security through obscurity" by incorporating multiple "layers." The layers are used to provide extra protection because an attacker can only get to the next layer if he successfully decrypted the previous layer. This is very valuable because it prevents one from opening the code and removing the header with little to no challenge. The SafeNet Sentinel has the option to set this from one layer to five layers, with three being set as the default.

The Aladdin HASP HL implements the use of 128-bit AES encryption by encrypting the program.

The Aladdin HASP HL also implements RSA, but this is only for the Remote Update.

Aladdin's HASP HL incorporates a version of program protection using what they refer to as an "envelope," which consists of 128-bit AES encryption and 'security through obscurity' by using "Anti-Debugging and Reverse Engineering Modules." The layers are used to provide extra protection because an attacker can only get to the following layer if the current layer has been successfully decrypted. The Aladdin HASP HL offers the option to set the number of layers from one to fifty layers with a default set to twelve layers.

Experiments performed

Given the complexity of the encryption algorithms used, brute force decryption was not an option. Attempting such a method would require an enormous amount of time; it was not a feasible option for someone attempting to compromise the security of this protection. That said, the following experiments were performed: compare the way in which each product implements the use of a wrapper to protect the program, compare the way the protected program interacts with the USB key and can the protected program be tricked into working by using a replay attack.

Experiment 1: Evaluation of Wrapper Protection

Description

The protection of a programs physical code is a good way to provide protection against a malicious user. To test the SafeNet Sentinel and Aladdin HASP HL's ability to protect a program, the areas of wrapper strength and implementation as well as the encryption algorithms used to protect the program will be tested.

Part A) Comparison of wrapper strength - Size

Details

Since the actual design of the system is closed-source, the researchers were only able to discover the features of the two different hardware keys by way of company documentation – Aladdin's Programmer's Guide, and SafeNet's Developer's Guide documentation.

Experimental Design

In an attempt to examine more closely the strength of each of the wrappers, four applications were tested. These applications were:

md5sum.exe (http://theopencd.sunsite.dk/md5.php)
putty.exe version 0.58
(http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html)
SnoopyPro.exe version 0.22
(http://sourceforge.net/project/showfiles.php?group_id=34567)
a2.exe (custom built 'Hello World' application).

The process for testing the strength of each application consisted of noting the date and time as well as the file size for each of the files. Following this, each application was wrapped with both the SafeNet and Aladdin wrapping applications using the default number of layers.

Data

HASP HL

- Currently there is a range of protective layers for the HASP key
    - Ranges from 1 to 50 in value
    - Default is 12

Figure 9.1 – HASP Shell notes

Sentinel

- Has a range of levels for the shell (1 to 5)
    - Default is 3
- Works by using a multi layering system
    - The current layer can only be decrypted if the previous layer was successfully decrypted.

Figure 9.2 – Sentinel Shell notes

Below is a table summarizing the results, all sizes are listed in bytes:

Aladdin HASP HL, Layer 12 of 50:

Program Name		Unwrapped	Wrapped	Wrapper Size
md5sum.exe	Size	49,152	1,531,904	1,482,752
	Disk	49,152	1,531,904	1,482,752
putty.exe	Size	421,888	1,904,640	1,482,752
	Disk	421,888	1,904,640	1,482,752
SnoopyPro.exe	Size	749,568	2,232,320	1,482,752
	Disk	749,568	2,232,320	1,482,752
a2.exe	Size	8,873	1,499,817	1,490,944
	Disk	12,288	1,503,232	1,490,944

SafeNet Sentinel, Layer 3 of 5:

Program Name		Unwrapped	Wrapped	Wrapper Size
md5sum.exe	Size	49,152	3,426,184	3,377,032
	Disk	49,152	3,428,352	3,379,200
putty.exe	Size	421,888	3,802,840	3,380,952
	Disk	421,888	3,805,184	3,383,296
SnoopyPro.exe	Size	749,568	4,134,584	3,385,016
	Disk	749,568	4,136,960	3,387,392
a2.exe	Size	8,873	3,379,153	3,370,280
	Disk	12,288	3,379,200	3,366,912

Results

Aladdin

Based upon these experiments, the date and time were found to be insignificant. However, in three of the four applications wrapped with Aladdin, the wrapper size remained the same. This was a point of concern that led to the next experiment.

SafeNet

Based upon these experiments, it was found that sizes varied each time that the same application was wrapped. It is believed that time may have been a factor, but there was nothing substantial to prove this theory. The wrapped application size varied by a relatively small amount (the results were all within a 200-byte range). In addition, the wrapper size for each of the applications varied.

Part B) Comparison of wrapper strength – Susceptibility to Attack

Details

The wrapper that is added to the program is one of the main areas of protection that is offered by each product. Because of this importance, both the SafeNet Sentinel and Aladdin HASP HL protected programs were opened with a decompiler in attempts to discover useful information or even bypass the security mechanisms that have been put in place.

Experimental Design

A very simple and small application (ex, "Hello World") was created that both programs could wrap successfully. Following the wrapping of the application, attempt to decompile the program and investigate. Upon investigation, attempts were made to strip off the hardware-based protection calls as well as gather potentially valuable information about the program itself.

Using IDA Pro 5 (Demo Version, http://www.datarescue.com), the program was decompiled into assembly language and a graphical display of the program was presented.

Data

The Aladdin wrapped program yielded four execution boxes with a payload of information that could not be understood by the program or the researchers. However, it was consistent with Aladdin's documentation on their Envelope. Upon further investigation, the code that decides whether a HASP user key is attached was discovered.

By inserting a breakpoint (a place to stop executing the program), the specific assembly command that passes or fails and then displays the error message "HASP key not found" was discovered.

The Sentinel wrapped program yielded what could be described as a multi-tiered program structure, several lines of code with several gaps in the code. The specific assembly code that queries if the Sentinel key is attached could not be found.

Results

SafeNet Results
Because of SafeNet Sentinel's apparent code-obfuscation, attempts at breaking down the code and stripping off the hardware checking mechanism were unsuccessful.

Aladdin Results
Even though the Aladdin HASP HL also incorporates the use of code obfuscation, some information was able to be gathered from the Aladdin HASP envelope. However, it was not enough to pose a serious danger to the security of the program. In the opinion of the researchers, this is a point of concern and should be addressed as soon as possible.

It is important to note that both of these tests were conducted without the use of the API functions offered by both companies. By incorporating both the wrapping as well as the API functions, one can ensure the highest level of security for the protected program.

SafeNet Sentinel:	100 out of 100
Aladdin HASP HL:	90 out of 100

Experiment 2: Program – USB key interaction

Description

In this experiment, the way in which data is sent to and from the host computer to the USB key will be examined. This is an area that a potential attacker could use to gather data and attempt an exploit.

Experimental Design

The program "Notepad.exe" was wrapped using both the SafeNet and Aladdin wrappers and the option to allow debugging tools was enabled. Once both programs were successfully wrapped, a USB sniffing program was started, in this case, it was HHD Software's USB Monitor version 2.37, and then the wrapped applications were executed. With the collected data, the researchers would attempt to reconstruct and/or extract any useful information.

Data

Using a USB sniffer, the interaction between the program and the USB key could be documented.

The SafeNet Sentinel appeared to follow a routine of:
'a, b' are some numbers
USB key: Send a bytes (to the USB key)
USB key: Get b bytes (from the USB key)

The Aladdin HASP HL appeared to follow a routine of:
'a, b, c' are some numbers
USB key: Send a bytes
Host: Sending a bytes
    Setup packet: 40 1E xx xx xx xx [1-3]0 00

USB key: Get b bytes
Host: Getting c bytes
    Setup packet: C0 9E 00 00 00 00 12 00

There was some information that was gathered about the packet structure of the Aladdin HASP HL. With the setup packet that was sent initially, (40 1E …), the second last group, [1-3]0, was determined to be a length value of the packet. There appeared to be only values of 10, 20, and 30 bytes.

In the second setup packet, (C0 9E …), the second last group, 12, was also determined to be a length value of the packet. The notable thing about this is that every time this packet was 12 bytes.

SafeNet Sentinel

```
000033: Bulk or Interrupt Transfer (UP), 19.05.2006 16:17:31.7812500
+0.0
Pipe Handle: 0x81fda4b4 (Endpoint Address: 0x1)
Send 0x40 bytes to the device:
  14 49 7D 00 40 0F 12 57 E7 AF 89 7D 9C 37 9C 49    .I}.@..Wç¯‰}œ7œI
  61 48 CE 71 AF 33 20 93 3E 3F 7A 31 64 B7 08 98    aHÎq¯3 "›?z1d·.˜
  60 75 F9 66 8E 85 37 D0 00 A1 28 38 17 93 89 68    `uùfŽ…7Ð.¡(8."‰h
  63 8E 47 B2 CF 1C F0 34 CF 30 D0 70 97 E1 DF 4D    cŽG²Ï.ð4Ï0Ðp—áßM
```

```
000034: Bulk or Interrupt Transfer (UP), 19.05.2006 16:17:31.7968750
+0.0156250
Pipe Handle: 0x81fda4d4 (Endpoint Address: 0x81)
Get 0x23 bytes from the device:
  14 FF 02 9B C6 0D 7A F3 FF 19 EB 97 A7 43 C8 E7    .ÿ.›Æ.zóÿ.ë—§CÈç
  56 0B 0C 8A 65 96 AE 6A E3 93 E0 CB F6 EF E8 28    V..Še–®jã"àËöïè(
  E3 82 A5                                           ã‚¥
```

```
000035: Bulk or Interrupt Transfer (UP), 19.05.2006 16:17:31.7968750
+0.0
Pipe Handle: 0x81fda4b4 (Endpoint Address: 0x1)
Send 0x40 bytes to the device:
```

```
  15 48 7C 00 40 0F 3E 94 F3 B6 C4 F2 56 C6 72 5B    .H|.@.>”ó¶ÄòVÆr[
  EA 82 41 77 A4 5A C3 7D 90 19 90 CC 5D 7D 00 5E    ê,Aw¤ZÃ}□.□Ì]}.^
  1D 81 94 FA 11 00 35 1B 91 27 85 F3 0D 64 39 94    .□”ú..5.‘'…ó.d9”
  17 FD CA 14 F5 5C 22 5E AE DF 02 F0 12 91 D7 3A    .ýÊ.õ\"^®ß.ð.‘×:
```

```
000036: Bulk or Interrupt Transfer (UP), 19.05.2006 16:17:31.7968750
+0.0
Pipe Handle: 0x81fda4d4 (Endpoint Address: 0x81)
Get 0x23 bytes from the device:
  15 FF 02 D0 C4 39 61 58 6E F7 10 C3 A4 B6 0A 14    .ÿ.ÐÄ9aXn÷.Ã¤¶..
  77 B1 18 12 91 00 E6 7D F7 42 EC E0 95 1C 79 C8    w±..‘.æ}÷Bìà•.yÈ
  96 B9 53                                           –¹S
```

This is a packet capture of the USB traffic using SafeNet Sentinel.

Aladdin HASP HL

```
000093: Vendor-Specific Request (DOWN), 19.05.2006 16:31:13.8906250 +0.0
Destination: Device
Reserved Bits: 0
Request: 0x1e
Value: 0xa83
Send 0x20 bytes to the device

000094: Control Transfer (UP), 19.05.2006 16:31:13.9062500 +0.0156250
Pipe Handle: 0x823dcf80
  31 09 1D 41 5E F6 DA 46 5F 96 CB 17 DE 5B 62 7F    1..A^öÚF_–Ë.Þ[b□
  80 C0 0E 58 21 B7 A9 A3 5F 2C 32 F5 F9 25 67 82    €À.X!·©£_,2õù%g,

                                          Setup Packet

000095: Vendor-Specific Request (DOWN), 19.05.2006 16:31:13.9062500 +0.0
Destination: Device
Reserved Bits: 0
Request: 0x9e
Value: 0x0
Get 0x12 bytes from the device

000096: Control Transfer (UP), 19.05.2006 16:31:13.9062500 +0.0
Pipe Handle: 0x823dcf80
```

15 F8 C8 62 20 AD 6A 81 94 8D 5D B9 68 F8 7A 28 8C 54	.øÈb j□"□]¹høz(Œ T

This is a packet capture of the USB traffic using Aladdin HASP HL.

Results

Both products were able to protect the transmission of data between the USB key and the protected program. Even though data was gathered, it proved to be of little value because the data appeared to be encrypted.

Both products implement 128-bit AES encryption, which is strong by itself; however, the use of multiple encryption algorithms allows for a higher level of security to be reached.

SafeNet Sentinel:	100 out of 100
Aladdin HASP HL:	90 out of 100

Experiment 3: Replay Attacks

Description

A replay attack is an attack on the hardware based software protection which 'tricks' the program into thinking the key is present. This is accomplished by using tools that can record the traffic that is sent to a USB device. This usually will only work when the data that is sent to and from the USB key is static.

Experimental Design

The program "Notepad.exe" was wrapped using both the SafeNet and Aladdin wrappers and the option to allow debugging tools was enabled. Once both programs were successfully wrapped, a USB sniffing and recording program was started (HHD Software's USB Monitor version 2.37), and then the wrapped applications were executed. While the application was opening, the traffic that was generated was recorded. Following the recording of the traffic generated, the USB key was removed from the computer and the log file was replayed as the program was opened.

Results

While attempting to do a replay attack, both programs successfully denied the use of the program without the actual key in place.

SafeNet Sentinel:	100 out of 100
Aladdin HASP HL:	100 out of 100

Scoring

Security	**Aladdin** HASP HL	**SafeNet** Sentinel Hardware Key
Evaluation of Wrapper Protection	90	100
USB Key Interaction	90	100
Replay Attacks	100	100
Overall	**93**	**100**

The scoring reflects SafeNet's implementation of apparent code obfuscation in its file structure as well as its implementation of multiple layers of encryption to secure the traffic between the USB key and the program. Neither program failed but there were some features that made SafeNet's Sentinel stand out.

References

Tulloch, Mitch. Microsoft Encyclopedia of Security. Microsoft Press, 2003.

http://research.microsoft.com/~klauter/IEEEfinal.pdf
Contains information about ECC and RSA equivalent encryption methods.

Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS). http://www.ietf.org/rfc/rfc3278.txt.

Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS). http://www.ietf.org/rfc/rfc3565.txt.

Aladdin HASP Programmers Guide version 12

SafeNet Developer's Guide version 1.0

Aladdin Website - http://www.aladdin.com/HASP/default.asp

SafeNet Website - http://www.safenet-inc.com/

Section 10: Licensing Models

Introduction

In this section, the licensing models of both the Aladdin HASP HL and the SafeNet Sentinel were examined.

Features

SafeNet offers several keys with varying configurations that allow for a variety of licensing schemes. The key models considered for this section were the S, SN, ST, and SNT. The S key offers standalone non-real time clock licensing. The SN offers networked non-real time clock licensing. The ST offers standalone licensing with an on-board real time clock. The SNT offers networked licensing with an on-board real time clock.

Aladdin offers several keys with varying configurations that allow for a variety of licensing schemes. The key models considered for this section were the HASP4 Standard, HASP4 M1, HASP4 M4, HASP4 Time, and HASP4 Net. The HASP4 Standard offers no internal read/write capabilities and only basic HASP protection. The HASP4 M1 and M4 models differ only in their amount of on-board memory and the number of applications they can protect. The HASP4 Time offers an on-board real time clock and protection for up to eight applications. The HASP4 Net comes in several variations based on the number of workstations, and in an unlimited model, and offers network based licensing.

Experiments to be performed

The licensing models were reviewed in several areas: limitations to the number of keys which can be programmed by the programmer/distributor key; the number of applications which can be protected by one single key; the ability to create a custom license; and the ability to activate or deactivate licenses in the field. Finally, the team considered if the use of any programming language limited license flexibility and if a relationship existed between API and licensing. Each category was initially assigned a total score of 100 out of 100 points. Five points were then deducted for each perceived failure. A final score was then averaged from the scores of all reviewers.

Interpretation of results

The team reviewed the HASP programmers guide version 12, the SafeNet developers guide version 1, as well as the websites, and customer support lines of each company.

Experiment 1: Number of Keys that can be Programmed

Description
The research team determined the number of keys that could be programmed by the distributor/programmer key. Documentation was reviewed and representatives from each company were contacted.

Data/Results
HASP: 100/100
Aladdin allows the programmer to program an unlimited number of keys.

SafeNet: 100/100
SafeNet allows the programmer or distributor to program an unlimited number of keys.

Experiment 2: Number of Applications per User Key

Description
The research team determined the number of applications that could be protected by a single end user key. Documentation was reviewed and representatives from each company were contacted.

Data/Results
HASP: 100/100
When the research team contacted the Aladdin offices, it was told the following:

- The HASP HL Max can protect up to 112 applications;
- The HASP HL Pro Key can protect up to 16 applications;
- All other HASP HL keys can protect up to 12 applications;
- The HASP HL Standard can only protect one application.

Sentinel: 100/100
The SafeNet Sentinel can protect up to 112 applications, but the exact number depends on the licensing model desired and the amount of memory that is consumed by the particular model.

Experiment 3: Is There a Way to Create a Custom License?

Description
The research team determined the flexibility in the provided licensing models for each key. The provided documentation for each key was reviewed for licensing options.

Data/Results

HASP: 100/100

Aladdin has the ability to use both time and execution based licensing allowing for a high degree of flexibility in trial licensing. Additionally, the ability of HASPEdit HASP4 Net keys allows vendors to tune their licensing systems for their customers' needs.

Sentinel: 100/100

SafeNet has the ability to use both time and execution based licensing allowing for a high degree of flexibility in trial licensing. Additionally, the ability to set soft limits in the ST and SNT for the number of concurrent executions in network based licensing allows vendors to tune their licensing systems for their customers' needs.

Experiment 4: License Activation/Deactivation in field

Description

The research team determined the ability of both systems to be activated and terminated after reaching the end-user. The provided documentation for each key was reviewed for licensing options.

Data/Results

HASP: 100/100

Sentinel: 100/100

Both Aladdin and SafeNet offered the ability to update their keys remotely, which allows a vendor to enable a key once it reaches an end user. Additionally, a correctly configured key can be set to expire after a certain date or number of executions, which allows a vendor to control how their software is used.

Experiment 5: API and Licensing Model Correlation

Description

The research team determined the relationship between licensing and API. The provided documentation for each key was reviewed for the way in which the license may be implemented in the API.

Data/Results

HASP: 100/100

Sentinel: 100/100

Since the API controls any special functionality that the developer wants to integrate for the user, the API has only a correlation to the general API beyond a "trial mode" function that may take place with the other types of licensing discussed above. The API would be the control for licensing layered control and advanced functionality at the general use layer.

Analysis

Both HASP and SafeNet offer a variety of options with their licensing. Both heavily utilize the API to allow for granular control. Generic trial modes were created by both HASP and SafeNet via a limitation on the number of executions or the amount of time the key could run before it would stop working.

Summary

After analyzing all factors for licensing as presented by both SafeNet and Aladdin, the only major difference that was discovered is HASP's lack of a virtual clock. Both companies offer a variety of different licenses that allow the vendor to distribute its program with equal flexibility.

Scoring

Licensing Models	**Aladdin** HASP HL	**SafeNet** Sentinel Hardware Key
Number of Keys that can be Programmed	100	100
Number of Applications Per User Key	100	100
Is There a Way to Create a Custom License?	100	100
License Activation/Deactivation in the Field	100	100
API and Licensing Model Correlation	100	100
Overall	**100**	**100**

SafeNet scores a 100 out of 100 points for this section, and HASP scores a 95 out of 100 points for this section since they do not implement a virtual clock.

References

Aladdin HASP Programmers Guide version 12
SafeNet Developer's Guide version 1.0
Aladdin Website - http://www.aladdin.com/HASP/default.asp
SafeNet Website - http://www.safenet-inc.com/

Section 11: Protection Features

Introduction

The protection features offered by the Aladdin HASP HL and SafeNet Sentinel hardware keys deal with the types of files each can protect. This section was addressed by performing side-by-side experiments with both products and examining the results. These experiments were performed in order to determine the extent of comparable protection features offered by each product.

Features

According to the documentation for the Aladdin HASP HL and SafeNet Sentinel, support exists for the wrapping of Win32 PE's, Win32 PE DLL's, COM DLL's, .NET Executables (v1.0, v1.1).

Experiments performed

The experiments performed included testing the ability of each product to protect Windows 32-bit portable executables, Windows 32-bit PE DLL's, and the various versions of Microsoft .NET framework (v1.0, 1.1, 2.0). In addition, the protected application's functionality when the key is removed after starting the applications was tested.

The above experiments were performed side-by-side on identical systems using comparable vendor tools and processes. The tests were designed to find out what each product was able to protect, and did not take in to account the steps needed or the ease of use. The experiments were performed following the instructions included with each product, as well as using the appropriate help files. Only the features that were common to both products were compared based on the extent to which they could secure each type of file. Each experiment was also performed three times with each product to ensure consistency.

Interpretation of results

During the course of each experiment, the investigators recorded which products were able to fulfill certain protection requirements and rated them in comparison to each other on a scale of 1-100. For example, if one product was able to protect one of two items, and the other was able to protect two of two items, the first one would get a rating of 50 while the second would get a rating of 100.

Experiment 1: 32-bit Portable Applications

Using two identical systems, the experiment protected Windows 32-bit portable executables and tried executing them after protection, with and without a key.

This experiment was performed using the Adobe Acrobat Reader executable (Version 7.0.0), AcroRd32.exe. This was used because it is a widely available and familiar program. The appropriate vendor tools were used to wrap the executable. After

protecting the executable, the investigators attempted to run the executable with and without the key. It was expected that the executable would fail to start without the key present, and that a message would be shown to the user indicating that the key was not present. It was also expected that the executable would successfully start with the key present, and would not show any error messages.

Data/Results

HASP HL

This product successfully wrapped the executable without any errors. Upon attempting to run the executable without the key present, an error message was shown indicating that the key was not present, and the executable failed to start, as expected. Upon attempting to run the executable with the key present, no errors were shown and the executable started successfully, as expected. The experiment yielded the same results on all three attempts.

Two aspects of the experiment:
Wrapped Win32 PE – Successful
Win32 PE Execution Protection – Successful

Rating: 100/100

SafeNet Sentinel

This product successfully wrapped the executable without any errors. Upon attempting to run the executable without the key present, an error message was shown indicating that the key was not present, and the executable failed to start, as expected. Upon attempting to run the executable with the key present, no errors were shown and the executable started successfully, as expected. The experiment yielded the same results on all three attempts.

Two aspects of the experiment:
Wrapped Win32 PE – Successful
Win32 PE Execution Protection – Successful

Rating: 100/100

Experiment 2: Windows DLLs

Using two identical systems, the experiment protected Windows DLLs and tried executing the appropriate application, after protection, with and without a key.

This experiment was performed using the Adobe Acrobat Reader core DLL, AcroRd32.dll (Version 7.0.0.1333), as it was determined to be the easiest DLL to utilize in testing. The appropriate vendor tools were used to wrap the DLL. After protecting the DLL, the researchers attempted to run Adobe Acrobat Reader with and without the key. It was expected that the program would fail at some point without the key present, and that a message would be shown to the user indicating that the key was not present. It was also expected that the executable would successfully start with the key present, and would not show any messages or errors.

Data/Results

HASP

This product successfully wrapped the DLL without any errors. Upon attempting to run the application without the key present, an error message was shown indicating that the key was not present, and the application failed to start, as expected. A second message was also shown, indicating that Adobe Acrobat Reader was unable to access its core DLL file. Upon attempting to run the executable with the key present, no errors were shown and the executable started successfully, as expected. The experiment yielded the same results on all three performances.

Two aspects of the experiment:
Wrapped Win32 PE DLL – Successful
Associated application functioning properly – Successful

Rating: 100/100

Sentinel

This product successfully wrapped the DLL without any errors. Upon attempting to run the application without the key present, an error message was shown indicating that the key was not present, and the application failed to start, as expected. A second message was also shown, indicating that Adobe Acrobat Reader was unable to access its core DLL file. Upon attempting to run the executable with the key present, no errors regarding the key were shown, but Adobe Acrobat Reader still reported being unable to load its core DLL file, and the application failed to start. In the opinion of the investigators, this product is unable to wrap some DLL files without corrupting them, in the testing environment. The experiment yielded the same results on all three performances.

Two aspects of the experiment:
Wrapped Win32 PE DLL – Successful
Associated application functioning properly – Failed

Rating: 50/100

Experiment 3: .NET Testing

Using two identical systems, the experiment protected Microsoft .NET applications and determined what level of .NET framework each product was able to protect.

This experiment was performed using a simple bouncing ball application built on the .NET framework. The appropriate vendor tools were used to wrap the application. After protecting the application, the investigators attempted to run it with and without the key. It was expected that the application would fail at some point without the key present, and that a message would be shown to the user indicating that the key was not present. It was also expected that the application would successfully start with the key present, and would not show any messages or errors. The experiment was performed three times each for .NET framework 1.0, 1.1, and 2.0.

Data/Results

HASP

This product successfully wrapped the application without any errors, as long as the .NET framework 1.0 or 1.1 was used. When the researchers attempted to wrap the application while using .NET framework 2.0, the vendor tools used to wrap the program showed error messages indicating that the .NET framework being used was not supported. The remainders of the experiments were carried out without using .NET 2.0 framework. Upon attempting to run the application without the key present, an error message was shown indicating that the key was not present, and the application failed to start, as expected. The experiment yielded the same results on all three performances.

Six aspects of the experiment (5 included in rating due to 1 being not testable):
Wrapped .NET 1.0 application – Successful
Wrapped .NET 1.1 application – Successful
Wrapped .NET 2.0 application – Failed
.NET 1.0 Application Execution Protection – Successful
.NET 1.1 Application Execution Protection – Successful
.NET 2.0 Application Execution Protection – N / A (not included in the scoring)

Rating: 80/100

Sentinel
This product successfully wrapped the application without any errors, on all versions of the .NET framework. Upon attempting to run the application without the key present, an error message was shown indicating that the key was not present, and the application failed to start, as expected. Upon attempting to run the executable with the key present, no errors were shown and the executable started successfully, as expected. The experiment yielded the same results on all three performances with all three versions of the .NET framework.

Six aspects of the experiment:
Wrapped .NET 1.0 application – Successful
Wrapped .NET 1.1 application – Successful
Wrapped .NET 2.0 application – Successful
.NET 1.0 Application Execution Protection – Successful
.NET 1.1 Application Execution Protection – Successful
.NET 2.0 Application Execution Protection – Successful

Rating: 100/100

Experiment 4: Removing Key during Application Execution

Using two identical systems, the experimenter attempted to discover if removing the key after starting a protected application had an effect on the application's functionality.

During the previous three experiments, the key was removed after successfully starting the application. This was done to find the extent of protection offered by each product when the key is removed, and the default settings were used with each product. Although one product had customizable settings for this feature, the fact that the feature worked in each product is what was tested.

Data/Results

HASP
This product has a customizable setting that can be adjusted to specify how often the key should be checked for presence in the system. This setting was left at its default of 5 minutes. After starting the application and removing the key in each experiment, the investigators timed how long it took before seeing a message indicating the absence of the key, and found that after 5 minutes the application always stopped until the key was reinserted. The experiment yielded the same results on all three performances.

Four aspects of the experiment:
Win32 PE key removal protection – Successful
Win32 PE DLL key removal protection – Successful
.NET framework 1.0 key removal protection – Successful
.NET framework 1.1 key removal protection - Successful

Rating: 100/100

Sentinel
This product did not have a customizable setting that could be adjusted to specify how often the key should be checked for presence in the system. After starting the application and removing the key in each experiment, the investigators timed how long it took before seeing a message indicating the absence of the key, and found that after an average of 1.5 minutes, the application always stopped until the key was reinserted. The experiment yielded the same results on all three performances.

Five aspects of the experiment:
Win32 PE key removal protection – Successful
Win32 PE DLL key removal protection – Successful
.NET framework 1.0 key removal protection – Successful
.NET framework 1.1 key removal protection – Successful
.NET framework 2.0 key removal protection - Successful

Rating: 100/100

Summary

The experiments found that both products fared about the same in all of the experiments, regardless of the overall rating of each product, in the testing environment. Both products were able to protect Windows 32-bit portable executables without errors. Only the HASP HL was able to successfully protect a Windows 32-bit portable executable DLL file, while the Sentinel product seemed to corrupt the protected DLL. Both products were able to successfully protect applications that use the Microsoft .NET framework, versions 1.0 and 1.1, but only the SafeNet Sentinel was able to successfully protect .NET framework 2.0. It must be pointed out that HASP does not claim to be able to protect this version of .NET framework, while Sentinel does. Considering this, the investigators feel that both products meet their claims in this feature. Finally, both products offer protection against key removal, but at different levels. The investigators believe that being able to customize this feature, as in the case of HASP, is useful, but they also believe that the static setting of about 1.5 minutes with the Sentinel product is sufficient. Please see the tables below for the test results.

SafeNet Sentinel

Wrapped Win32 PE	Successful
Win32 PE Execution Protection	Successful
Wrapped Win32 PE DLL	Successful
Associated application functioning properly	Failed
Wrapped .NET 1.0 application	Successful
Wrapped .NET 1.1 application	Successful
Wrapped .NET 2.0 application	Successful
.NET 1.0 Application Execution Protection	Successful
.NET 1.1 Application Execution Protection	Successful

.NET 2.0 Application Execution Protection	Successful
Win32 PE key removal protection	Successful
Win32 PE DLL key removal protection	Successful
.NET framework 1.0 key removal protection	Successful
.NET framework 1.1 key removal protection	Successful
.NET framework 2.0 key removal protection	Successful

Aladdin HASP HL

Wrapped Win32 PE	Successful
Win32 PE Execution Protection	Successful
Wrapped Win32 PE DLL	Successful
Associated application functioning properly	Successful
Wrapped .NET 1.0 application	Successful
Wrapped .NET 1.1 application	Successful
Wrapped .NET 2.0 application	Failed
.NET 1.0 Application Execution Protection	Successful
.NET 1.1 Application Execution Protection	Successful
.NET 2.0 Application Execution Protection	N/A
Win32 PE key removal protection	Successful
Win32 PE DLL key removal protection	Successful
.NET framework 1.0 key removal protection	Successful
.NET framework 1.1 key removal protection	Successful

Scoring

Protection Features	**Aladdin** HASP HL	**SafeNet** Sentinel Hardware Key
32-bit Portable Applications	100	100
Windows DLLs	100	50
.NET Testing	80	100
Removing Key During Application Execution	100	100
Overall	**95**	**87.5**

By dividing 100 points over a specified number of aspects in each experiment, the investigators were able to determine overall ratings.

This average weighs each feature equally. Ultimately, the user should look at the results of each experiment and determine how to weigh each result in accordance with their needs.

References

Aladdin - Support & Downloads. 15 May 2006.

SafeNet-Inc - Hardware Keys. 15 May 2006.

Conclusion

Overall, both products performed well. Below is a list of score averages for each section, as well as an overall average for the entire report. For more detailed results, consult the scoring portion of each section.

Hardware Features	Aladdin HASP HL	SafeNet Sentinel Hardware Key
Overall	**100**	**100**

Documentation	Aladdin HASP HL	SafeNet Sentinel Hardware Key
Overall	**95.83**	**91.33**

Protection Architecture	Aladdin HASP HL	SafeNet Sentinel Hardware Key
Overall	**54.97**	**71.42**

Remote Updates	Aladdin HASP HL	SafeNet Sentinel Hardware Key
Overall	**93**	**90**

Drivers & Libraries	Aladdin HASP HL	SafeNet Sentinel Hardware Key
Overall	**100**	**46**

Customer & Developer Perspective	Aladdin HASP HL	SafeNet Sentinel Hardware Key
Overall	**94.2**	**81.9**

API	Aladdin HASP HL	SafeNet Sentinel Hardware Key
Overall	**75**	**80**

Vendor Tools	Aladdin HASP HL	SafeNet Sentinel Hardware Key
Overall	**82.75**	**86.25**

Security	Aladdin HASP HL	SafeNet Sentinel Hardware Key
Overall	**93**	**100**

Licensing Models	**Aladdin** HASP HL	**SafeNet** Sentinel Hardware Key
Overall	**100**	**100**

Protection Features	**Aladdin** HASP HL	**SafeNet** Sentinel Hardware Key
Overall	**95**	**87.5**

Overall Average	**Aladdin** HASP HL	**SafeNet** Sentinel Hardware Key
Overall Score (Sum of Each Section Score divided by 11)	**89.43**	**84.95**

Please note that these scores do not constitute an endorsement of either product, nor should they be considered an assurance of suitability for any project. Before purchasing or using any product, the developer should read company documentation and contact a vendor's representative to discuss his particular needs.

Appendix A – Terms

AES Encryption:

The Advanced Encryption Standard algorithm is based on *Rijndael.* "Rijndael is a symmetric block cipher which means that it encrypts one block of data at a time. It was developed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen, and its design is based on an earlier block cipher named Square. Both the block length and key length in Rijndael are variable, with possible block lengths of 128, 192, or 256 bits and key lengths of the same three values (both lengths can be extended further in multiples of 32 bits). Rijndael uses a series of rounds to transform plaintext blocks into ciphertext, with more rounds used for bigger block sizes and larger keys (the actual number of rounds is equal to six more than the larger of the key and block sizes). Each round combines substitutions, rotations, XOR operations, and mixing of columns in the state table." (Tulloch)

ECC Encryption:
The Elliptic Curve Cryptography algorithm uses points on the curves, which "comprise a structure called a group. These points are the values that are used in mathematical formulas for ECC encryption and decryption processes. The algorithm computes discrete logarithms of elliptic curves, which is different from calculating discrete logarithms in a finite field (which is what Diffie-Hellman uses).

"Some devices have limited processing capacity, storage, power supply, and bandwidth, such as wireless devices and cellular telephones. With these types of devices, efficiency of resource use is very important. ECC provides encryption functionality, requiring a smaller percentage of the resources needed by RSA and other algorithms, so it is used in these types of devices." (Harris)

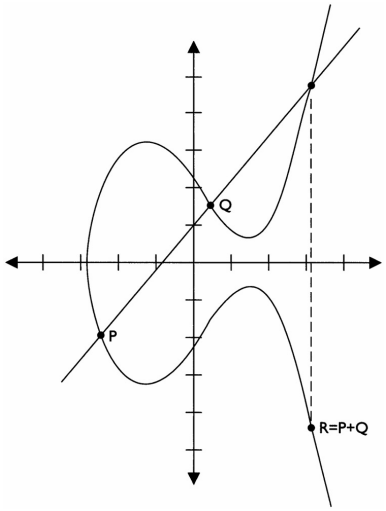


Figure 1.1 – This picture shows an example of an Elliptic Curve (Harris).

References

Tulloch, Mitch. Microsoft Encyclopedia of Security. Microsoft Press, 2003.

Harris, Shon. Cryptography CISSP: All-in-One Exam Guide. Third. McGraw-Hill/Osborne, 2005.

http://research.microsoft.com/~klauter/IEEEfinal.pdf
Contains information about ECC and RSA equivalent encryption methods.

Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS). http://www.ietf.org/rfc/rfc3278.txt.

Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS). http://www.ietf.org/rfc/rfc3565.txt.